Universidad
Carlos III de Madrid

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

PROYECTO FIN DE CARRERA

Sns_Xsens: La integración del sensor inercial MTi dentro de la arquitectura software YARP

Autor: Daniel García Sánchez
Tutor**:** Alberto Jardón Huete
Director: Juan Carlos González Víctores

Titulación**:** I.T.I. Electrónica Industrial

LEGANES, MADRID
DICIEMBRE 2010

AUTOR: DANIEL GARCÍA SÁNCHEZ

TUTOR: ALBERTO JARDÓN HUETE

DIRECTOR: JUAN CARLOS GONZÁLEZ VÍCTORES

Rama: I.T.I. ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

La defensa del presente Proyecto Fin de Carrera se realizó el día 10 de diciembre de 2010; siendo calificada por el siguiente tribunal:

Presidente: Ramón Barber Castaño

Secretario: Miguel González-Fierro

Vocal: Marta Portela Garcia

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente Secretario Vocal

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a Juan Carlos González Víctores por la ayuda durante la realización del proyecto.

Gracias a mí cuñado José por ayudarme cuando el proyecto se estancaba, a mi hermana Raquel por ayudarme con la estética de mi memoria, a mis padres y a mis hermanas por estar ahí para ayudarme en lo que fuera.

Gracia a mi amigo Alberto que nos hemos tenido que apoyar mutuamente con nuestros respectivos proyectos, a mis amigos de toda la vida Chema Borja y Raul y a mis amigos de la universidad por todos estos años de buenos momentos.

RESUMEN

El grupo Robotics Lab de la Universidad Carlos III de Madrid, desarrolló en 2004 un primer prototipo del robot asistencial ASIBOT, cuyo nombre en el contexto del proyecto europeo fue MATS (flexible Mechatronic Assistive Technology System to support persons with special needs in all their living and working environments), cuya finalidad es la ayuda y asistencia a personas discapacitadas y de la tercera edad.

Este proyecto fin de carrera surge al darse la necesidad de ampliar la gama de dispositivos de control y/o mando del robot asistencial ASIBOT.

Para esto se ha desarrollado dicho proyecto, cuyo objetivo específico es la integración del sensor acelerómetro e inclinómetro MTi de la marca Xsens. Su implementación específica consiste en la creación de un software envoltorio de los drivers de bajo nivel del sensor. El envoltorio ha sido programado en lenguaje C++, donde la comunicación entre el sensor y el robot asistencial ASIBOT o componentes software del mismo se realiza por medio de la arquitectura software YARP.

Como trabajo adicional, se ha desarrollado un sistema de transformación de datos de texto a una representación gráfica más amigable para desarrolladores de otros componentes. Este componente sirve además para cualquier módulo que cumpla con sus especificaciones.

ABSTRACT

In 2004, the Robotics Lab of Carlos III University developed the first prototype of the care robot ASIBOT, whose name in the European project was MATS (flexible Mechatronic Assistive Technology System to support persons with special needs in all their living and working environments). Its main purpose was to give aid and health care to handicapped and senior people.

This final project comes to expand the range of control devices of the care robot ASIBOT.

It has been developed in order to integrate the MTi accelerometer and inclinometer from Xsens. It implements a wrapper for the low level drivers. The wrapper has been developed in C++ language, and the communication between the sensor and the robot software components is performed under YARP software architecture.

Additionally, we have developed a graphic interface to transform textual data into a more developer-friendly graphic representation. This component is reusable by any other module which meets its specs.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En los últimos tiempos la esperanza de vida está aumentando de forma considerable. Por este hecho hace que parte de los desarrollos tecnológicos más avanzados estén encaminados a la mejora de la calidad de vida de estas personas. El desarrollo de sistemas robotizados dotados de un alto nivel de movilidad e inteligencia permitirá alcanzar estos objetivos [1].

Varios robots de este tipo fueron desarrollados en el pasado, sin embargo, se ha comprobado que tienen grandes limitaciones de uso en entornos domésticos dado que necesitan de grandes y complejos sistemas de control que hace muy difícil su transporte. Además, los robots están anclados permanentemente a una mesa o silla de ruedas limitando, de esta forma, su aplicación en diferentes habitaciones y dependencias domésticas.

Con todos estos datos, se creó el primer prototipo del robot asistencial ASIBOT mostrado en la figura 1, denominado inicialmente MATS, realizado por el grupo Robotics Lab, en el departamento de Sistemas y Automática de la Universidad Carlos III de Madrid [2]. El robot es capaz de adaptarse a diferentes entornos de la casa e inclusive desplazarse por la misma. El robot, por ejemplo, puede moverse por las paredes de una habitación, sobre el lavabo, estar anclado a la silla de ruedas y moverse con ella, etc. Para ello, la casa, debe estar equipada con un sencillo sistema de anclajes que sirve tanto para alimentar al robot como para dotarlo de apoyo [3].

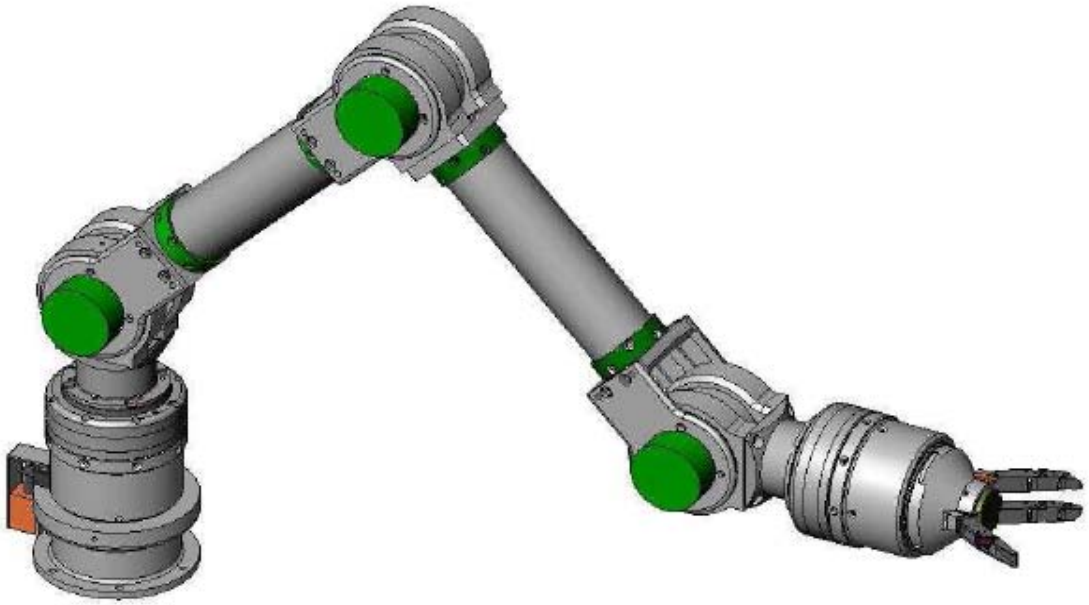

Figura 1: Robot asistencial ASIBOT anclado en un docking station

El uso y el manejo del robot debe ser fácil y amigable para poder ser usado por personas no técnicas y/o de una avanzada edad. Para poder disponer de una amplia gama de dispositivos de uso y manejo, el equipo encargado del desarrollo del robot ha implementado recientemente una infraestructura basada en módulos o componentes software para la integración de éstos, distribuidos a través de los equipos que forman el sistema de interacción y control del ASIBOT. La comunicación hombre-robot puede realizarse mediante distintas formas según las diferentes discapacidades del usuario: por voz, en caso de carecer de movilidad en las manos, por el manejo de un sencillo "joystick" en caso de discapacidades en los brazos, por un lápiz táctil, en el caso de discapacidad en las extremidades inferiores. El diálogo hombre-robot se efectúa mediante un sencillo sistema de menús orientados a la tarea basado en iconos gráficos [4]. A lo largo de este proyecto fin de carrera se detallará la integración del sensor acelerómetro e inclinómetro MTi de la marca Xsens.

1.1 Objetivos del proyecto

Este proyecto tiene como objetivo principal es crear la implementación necesaria para que el robot ASIBOT capte datos del sensor y para ello se han interpuesto los siguientes subobjetivos:

- Estudiar el sensor y las funciones que son proporcionadas para implementar el componente software.
- Diseñar e implementar el algoritmo para poder interactuar con el sensor y la captación de datos.
- Diseñar e implementar el algoritmo para poder enviar los datos al ASIBOT a través de la arquitectura software YARP.

Se ha implementado un componente software aparte, para poder ver los datos captados por el sensor en una gráfica y se le ha llamado Tol_visualize. De esta forma, los datos se ven representado de forma más amigable para usuarios y desarrolladores, y no sólo en modo texto mediante una terminal.

1.2 Estructura del documento

A continuación se describe la estructura de este proyecto, que se divide en cuatro capítulos.

En este primer capítulo se ha realizado una introducción explicando cuales han sido las motivaciones que han dado lugar a la realización del presente proyecto proponiendo para su ejecución una serie de objetivos a cumplimentar en el desarrollo del mismo.

En el segundo capítulo se describirá el ASIBOT, el sensor MTi de la marca Xsens®, la guía para componentes de robótica y las herramientas utilizadas.

En el tercer capítulo se explica cómo se hace la puesta en marcha, y se explica para qué sirven y cómo se han utilizado las distintas funciones que están a nuestra

disposición para la creación software del envoltorio y la visualización.

En el cuarto capítulo se explican las pruebas realizadas, las conclusiones finales y las posibles ampliaciones al presente proyecto.

2 Estado del arte

En este capítulo se habla del ASIBOT ya que la finalidad es integrar el sensor en su arquitectura. Se hablará del sensor MTi de Xsens tocando aspectos de firmware, comunicaciones y de sistemas de coordenadas utilizadas. Finalmente se hablará de las guías RCGv0.3 ya que éstas son las que se siguen durante la implementación de este módulo en la arquitectura software del ASIBOT.

2.1 ASIBOT

Si atendemos a la definición de robot escalador que propone el profesor Antonio Giménez [5] de la Universidad Carlos III de Madrid: “Un robot escalador es una máquina móvil que puede realizar tareas de servicio, de forma programada o autónoma, capaz de soportar su peso y propulsarse a sí mismo en su entorno de trabajo, que no tiene por qué ser horizontal. La máquina debe ser capaz de conseguir un alto nivel de sujeción, para que no se caiga ni deslice, en el caso de que se mueva sobre una superficie muy inclinada” y definiendo a los robots asistenciales como un tipo de robots de servicios especializados en asistir a personas discapacitadas o de avanzada edad podemos decir que el robot *ASIBOT* aúna diferentes características de distintas filosofías de diseño y, por lo tanto se le puede clasificar como un manipulador-escalador autoportado (debido a su reducido peso y tamaño).

Además puede montarse sobre una silla de ruedas cuando las circunstancias lo requieran, convirtiéndose así en un manipulador móvil; o que los dispositivos de agarre son externos al robot, los cuales además solucionan el problema que supone la

alimentación en la mayoría de los robot escaladores debido al cableado que limita la distancia a la que se puede desplazar o por el peso y tamaño que imponen las baterías requeridas.

El robot *ASIBOT* es simétrico y consta de cinco grados de libertad o ejes como puede verse en la figura 2.

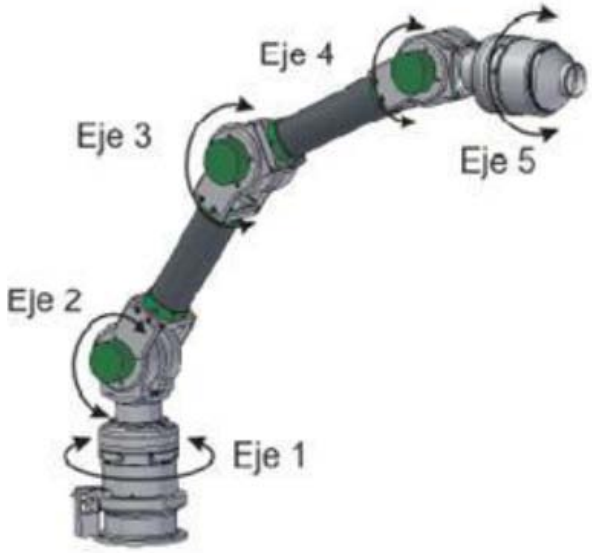

Figura 2: Distribución de las articulaciones en ASIBOT

Los dos eslabones centrales, a los cuales se les suelen llamar “brazos”, alojan en su interior el principal equipamiento electrónico y la unidad de control del robot. Los eslabones más pequeños, las “muñecas”, terminan en un mecanismo que le permite anclarse a los dispositivos de agarre. Estos elementos terminales poseen además una pinza con “dedos” que permiten coger objetos. Es importante señalar que los dispositivos de anclaje, o “*Docking Stations*”, proporcionan la alimentación que el robot necesita, por lo que el robot no tiene que cargar con ningún dispositivo de almacenamiento o generación de energía. La ausencia de fuente de energía, generalmente muy pesadas, unido a las materias primas empleadas, aluminio y fibra de carbono, le confiere al robot un peso aproximado de 13.5 Kg.

2.2 Xsens MTi

El MTi es un sensor de medición inercial con magnetómetros 3D integrados, con un procesador integrado capaz de calcular balance, cabeceo y guiñada en tiempo real, así como la salida calibrada 3D de aceleración lineal, velocidad angular (giroscopio) y los datos del campo magnético.

Nosotros utilizamos la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) correspondiente a la librería dinámica de C++ XsensCMT.DLL proporcionada por el proveedor del sensor. Al utilizar la librería de vínculos dinámicos (DLL por sus siglas en inglés) no es necesario tener duplicados de código objeto de la librería, con lo cual se reduce el tamaño en disco duro ocupado por los ejecutables que lo utilicen. Esto es la ventaja de la utilización de librerías dinámicas frente a la utilización de librerías estáticas.

2.2.1 Visión general de sistema MTi

En este apartado veremos de qué está compuesto internamente el sensor y explicaremos para qué sirve el filtro de Kalman de Xsens de 3 grados de libertad (XKF-3 por sus siglas en ingles).

2.2.1.1 Visión general

En la figura 3 se muestran los módulos de los que está compuesto el sensor por dentro y sus conexiones entre módulos.

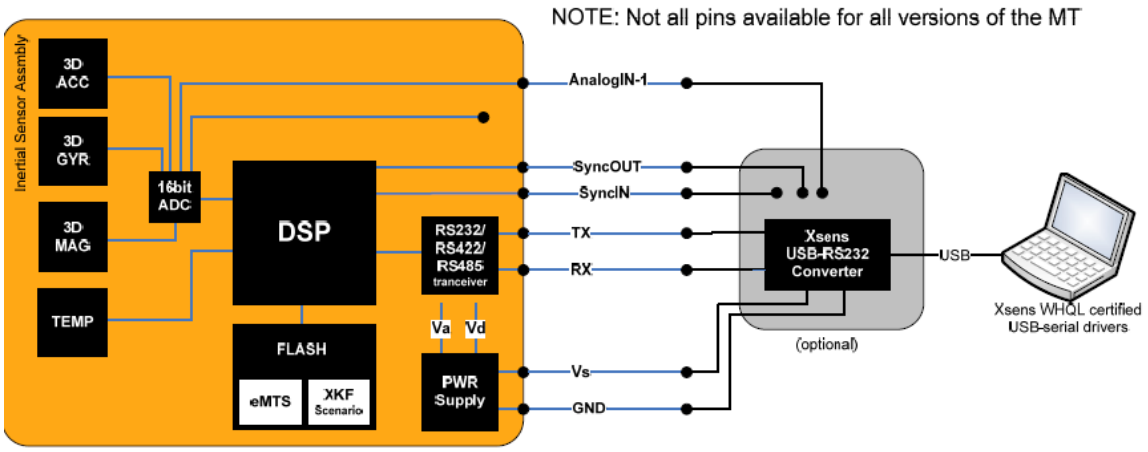

Figura 3: Visión en módulos del interior del sensor

2.2.1.2 Xsens Kalman Filter de MTi

La orientación del MTi se calcula por el filtro de Kalman de 3 grados de libertad (XKF-3 por sus siglas en ingles). XKF-3 utiliza las señales de los giroscopios, acelerómetros y magnetómetros para calcular una estimación estadística óptima de la orientación en 3D en alta precisión sin la interferencia de los desvíos tanto estáticos como dinámicos.

El diseño del algoritmo XKF-3 puede ser explicado como un algoritmo de fusión sensorial donde la medición de la gravedad (por los acelerómetros 3D) y el norte magnético de la Tierra (por los magnetómetros 3D) se compensan mutuamente lentamente, pero sin límites, incrementando (por deriva) errores de la integración de datos de tasa de giro. Este tipo de compensación de desvío a menudo se llama "referenciado por posición y rumbo", y a tal sistema a menudo se le llama "Attitude and Heading Reference System" (AHRS).

- USO DE LA ACELERACIÓN DE LA GRAVEDAD PARA ESTABILIZAR LA MEDIDA DE LA INCLINACIÓN

  XKF-3 estabiliza las medidas de inclinación utilizando las señales del acelerómetro. Un acelerómetro mide la aceleración de la gravedad, más la aceleración debida al movimiento del objeto con respecto a su entorno.

XKF-3 utiliza la hipótesis de que el promedio de la aceleración debida al movimiento es cero. Suponiendo esta hipótesis, la dirección de la gravedad puede ser observada y utilizada para estabilizar la medida de la posición. La orientación del MT en el campo de gravedad se explica de manera que las aceleraciones centrípetas o movimientos asimétricos no pueden causar un rendimiento estimado de orientación degradada. Esta suposición es sorprendentemente poderosa; casi todos los objetos que se mueven experimentan aceleraciones si se están moviendo, pero en la mayoría de los casos la aceleración media en lo que respecta al entorno durante un periodo de tiempo es igual a cero. La clave aquí es la cantidad de tiempo durante la cual debe ser la aceleración promedio para que la hipótesis sea cierta. Durante este tiempo, los giroscopios deben ser capaces de seguir la orientación con un alto grado de precisión. En la práctica, esto limita la cantidad de tiempo durante el cual el supuesto es cierto. Para los giroscopios utilizados en el MTi, este período de tiempo es de aproximadamente 10-20 segundos como máximo.

Sin embargo, para algunas aplicaciones, esta hipótesis no se sostiene. Por ejemplo, la aceleración de un automóvil puede generar aceleraciones significativas por períodos de tiempo que duran más que el tiempo máximo que los giroscopios pueden realizar un seguimiento fiable de la orientación. Esto seriamente degrada la precisión de las estimaciones de orientación con XKF-3, debido a que el escenario de uso (aplicación) no coincide con las hipótesis formuladas. Obsérvese, sin embargo, que tan pronto como el movimiento actúa de nuevo coincide con la hipótesis, XKF-3 se recuperará y se estabiliza. La recuperación de una precisión óptima puede llevar algún tiempo.

- USO DEL CAMPO MAGNETICO TERRESTRE PARA ESTABILIZAR EL HEADING (YAW)

  Por defecto, el “heading” está estabilizado usando el campo magnético local**,** el campo magnético medido se utiliza como una brújula. Si el

campo magnético de la Tierra local se encuentra temporalmente perturbado, XKF-3 hará un seguimiento de esta alteración en vez de asumir incorrectamente que no hay perturbación. Sin embargo, en caso de perturbaciones estructurales magnéticas (> 10 a 20 segundos), el "heading" calculará lentamente una solución con el nuevo norte magnético local. Tenga en cuenta que el campo magnético no tiene efecto directo en la estimación de inclinación.

En el caso especial de que el MTi esté rígidamente atado a un objeto que contiene materiales ferromagnéticos, entonces las perturbaciones magnéticas estarán presentes. El uso de un ***'magnetic field mapping'*** , hace que las perturbaciones magnéticas sean eliminadas, permitiendo que el MTi pueda ser utilizado como si no estuviera unido a un objeto que contiene materiales ferromagnéticos.

- INICIALIZACIÓN

  El algoritmo XKF-3 no sólo calcula la orientación, sino que también realiza un seguimiento de variables tales como las alteraciones de los sensores o las propiedades del campo magnético. Por esta razón, la salida de orientación puede llevar cierto tiempo estabilizarse una vez que el MT se pone en modo de medición. El tiempo para obtener una salida estable óptima depende de una serie de factores. Un factor importante que determina el tiempo de estabilización es determinado por el tiempo para corregir pequeños errores en el sesgo de la tasa de giroscopios. Estos errores son debidos a cambio de temperatura o la exposición a los impactos. Para reducir el tiempo de estabilización, el último error calculado puede ser almacenado en la unidad del sensor de memoria no volátil. Si el MTi se usa después de un corto período de tiempo al apagar el giroscopio, por lo general no ha cambiado mucho y el tiempo de estabilización será típicamente menos de 10 segundos. Por otra parte, XKF-3 se activara más rápido y llegará más rápido a la zona óptima de funcionamiento si se inicia en un entorno sin perturbaciones magnéticas.

2.2.2 Comunicación básica

Esta sección describe los conceptos básicos de cómo comunicarse con el MTi directamente en bajo nivel con comunicación RS-232 de serie, con o sin el uso de un convertidor de Xsens USB-serie.

El protocolo de comunicación, que es el mensaje de base, permite al usuario cambiar la configuración del MTi y recuperar los datos desde el dispositivo. La configuración es totalmente configurable por el usuario, por ejemplo, frecuencia de muestreo, y en la sincronización de la salida, velocidad de transmisión y modos de salida de datos, todo puede ser cambiado para adaptarse a tus necesidades.

Todos los cambios de configuración debe realizarse mientras el dispositivo está en el llamado estado de configuración. En este estado el dispositivo acepta los mensajes que ajusta el modo de producción o cambios en otros parámetros. Siempre que la configuración esta completada, el usuario puede configurar el dispositivo para el estado de medida. En este estado el dispositivo de salida de datos esta basados en la configuración actual.

2.2.2.1 Estados

El MTi tiene dos estados, es decir, Estado de configuración y el Estado de medición. En los ajustes de configuración de los distintos estados puede ser leído y escrito. En el Estado de medición, el dispositivo de salida de su mensaje de datos que contiene datos depende de la configuración actual.

Hay dos maneras de entrar en el Estado de configuración o el Estado de medición. En el encendido, el aparato empieza el procedimiento de atención, si no se toman medidas entrará en el estado de Medición por defecto, con su más reciente configuración almacenada.

Antes de entrar en el Estado de medición, el mensaje de configuración siempre se envía al controlador del sensor. Esta es la configuración que se lee desde la memoria interna no volátil y se utilizará en el estado de medición. Los datos en el mensaje de

configuración siempre puede utilizarse para determinar el modo de salida y los ajustes. También es posible entrar en el estado de configuración durante el encendido. Otra forma de entrar en el estado de configuración o del estado de medición consiste en utilizar el mensaje GoToConfig o GoToMeasurement respectivamente.

A continuación se muestra en la figura 4 una representación con los distintos estados posibles del MTi y las direcciones hacia donde se puede dirigir dependiendo del estado en el que se esté.

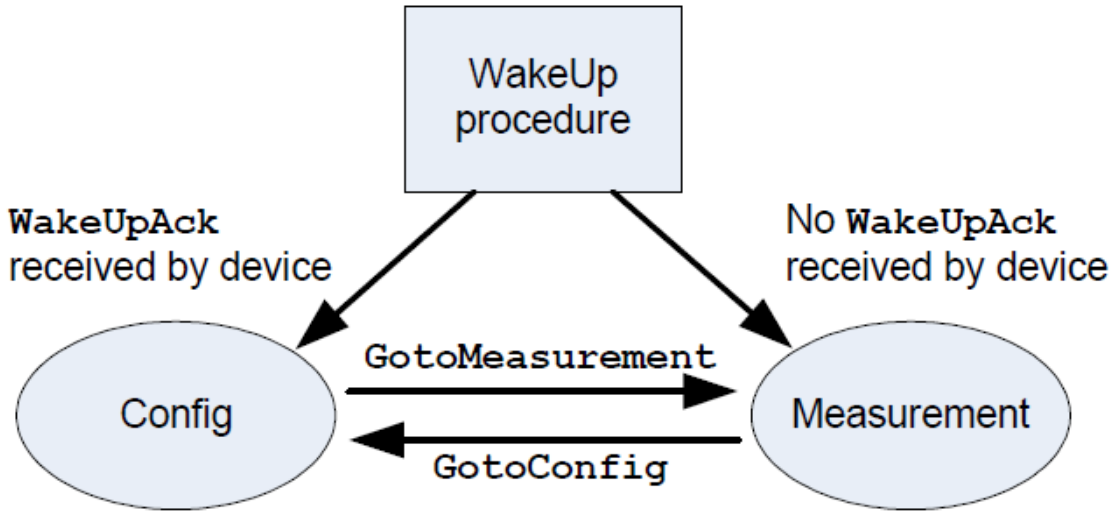

Figura 4: Estados del sensor

La configuración por defecto del MTi se muestra en la tabla siguiente.

Tabla 1: Configuración por defecto

Property	Value
Output mode	Orientation output
Output settings	Orientation in quaternion mode Sample counter enabled
Sample frequency	100 Hz
Baudrate	115k2 bps
Output skip factor	0

2.2.2.2 Mensaje

ESTRUCTURA DEL MENSAJE

La comunicación con el MTi se hace por medio de mensajes que se construyen de acuerdo a una estructura estándar. El mensaje MT estándar puede contener desde cero a 254 bytes de datos y la longitud total es de cinco a 259 bytes.

Un mensaje MTi contiene los siguientes campos:

Tabla 2: Campos de mensaje del MTi

Field	Field width	Description
PRE	1 byte	Preamble, indicator of start of packet → 250 (0xFA)
BID	1 byte	Bus identifier / address → 255 (0xFF)
MID	1 byte	Message identifier
LEN	1 byte	Value equals number of bytes in DATA field Maximum value is 254 (0xFE). Value 255 (0xFF) is reserved.
DATA	0 – 254 bytes	Data bytes (optional)
CS	1 byte	Checksum of message

Cabecera (PRE)

Cada mensaje comienza con la cabecera que siempre contiene el valor 250 (=0xFA **).**

Identificador de bus (BID) o la dirección

Todos los mensajes utilizados por el MTi utilizar el valor de la dirección 255 (0xFF) que indica un "dispositivo maestro". MT se utiliza en la Xbus tienen otras BID.

Identificador de mensaje (MID)

Este campo de mensaje identifica el tipo de mensaje.

Longitud (LEN)

Especifica el número de bytes de datos en el campo DATA. Valor 255 (= 0xFF) está reservado. Esto significa que un mensaje tiene una carga útil máxima de 254 bytes. Si la longitud es cero no existe campo de datos.

Datos (DATA)

Este campo contiene los bytes de datos y tiene una longitud variable que se especifica en el campo de longitud. La interpretación de los bytes de datos del mensaje depende del modo de salida.

Suma de comprobación (CS)

Este campo se utiliza para la comunicación de detección de errores. Si todos los mensajes bytes excluyendo la exposición de motivos se suman y el menor valor de bytes del resultado es igual a cero, el mensaje es válido y puede ser procesado. El valor de suma de comprobación del mensaje debe ser incluido en la suma.

En la figura 5 se muestra la composición del mensaje enviado por el MTi.

Figura 5: Composición del mensaje del MTi

2.2.2.3 Tiempo de comunicación

Para muchas aplicaciones que pueden ser cruciales para saber exactamente las diversas demoras y las latencias en el sistema. En esta sección se describe cómo el tiempo entre eventos físicos y el dispositivo de salida se relacionan en los modos de uso básico del MTi.

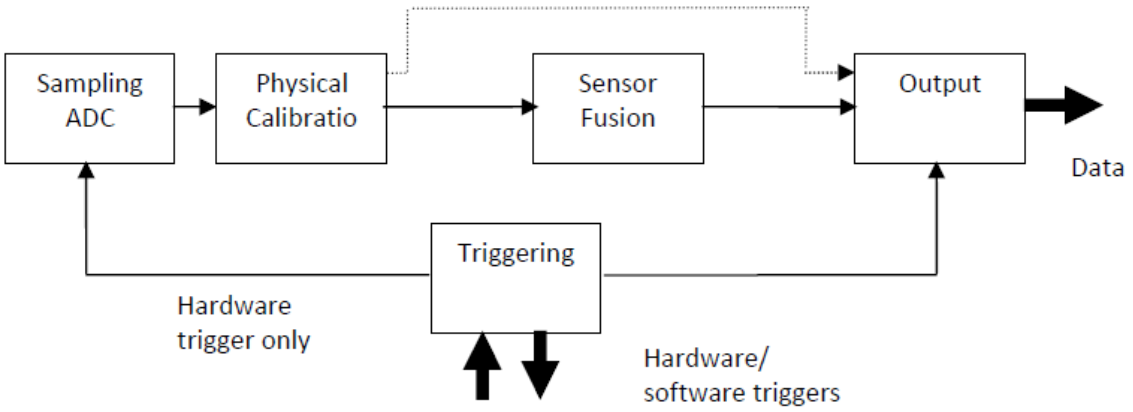

Figura 6: Bloques del sensor

Cuando el MTi está en medición de Estado, el DSP interno está continuamente ejecutando un bucle como el diagrama anterior. El disparo puede ser generado por muestreo del dispositivo interno, o de software externos, o incluso de hardware (por lo general no se recomienda).

El retraso de tiempo entre un evento físico (por ejemplo, un cambio de orientación o de aceleración) es dictado por dos factores: adquisición interna y tiempo de cálculo y tiempo de transmisión serie.

2.2.2.4 Disparo y sincronización

En caso de utilizar múltiples sistemas durante una medición, es importante contar con los datos de medición sincronizada entre los sistemas. El procesamiento de datos sincronizados es mucho más fácil porque no hay necesidad de volver a muestrear los datos para compensar las imprecisiones del reloj, como la deriva y las desviaciones del reloj. La sincronización con múltiples sistemas implica dos cuestiones importantes: el comienzo de la medición al mismo tiempo y la relación de tiempo fijo de las instancias de toma de muestras.

El MTi tiene capacidad para ser activado por dispositivos externos o accionar dispositivos externos para sincronizarse.

2.2.3 Especificación de salida

Hay varios modos de salidas diferentes del MTi. Los dos modos principales son la producción de Orientación y salida de datos calibrados. Éstos se examinan por separado. Sin embargo, tenga en cuenta que los dos modos de salida se pueden combinar fácilmente, de modo que se consigue un paquete de datos que tiene datos de orientación e inerciales calibrados, al mismo tiempo.

2.2.3.1 Sistema de coordenadas

Para la captación de datos el sensor tiene que tener claro los sistemas de coordenadas tanto del sensor como el de referencia por eso se explican a continuación.

- CALIBRADO DEL SENSOR

  Todas las lecturas del sensor calibrado (aceleraciones, velocidad de giro, el campo magnético de la tierra) están en el sistema cartesiano de coordenadas tal como se define en la figura 7. Este sistema de coordenadas es fijo al cuerpo del aparato, y se define como el sistema de coordenadas del sensor (S). La salida de la orientación 3D se discute más adelante.

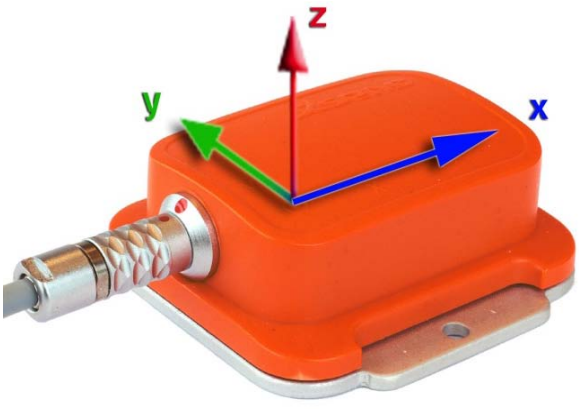

Figura 7: Sensor MTi con su sistema de coordenadas fijo (S)

La carcasa externa y la placa base de aluminio del MTi está cuidadosamente alineada con la salida del sistema de coordenadas. La alineación de la placa de fondo y los lados de la base de aluminio de la placa con respecto a la salida del sensor, es de 0,1 ° del sistema de coordenadas fijo (S).

La alineación entre la carcasa (de plástico) y el sistema de coordenadas fijo (S) de la salida del sensor, no es de alta precisión. La alineación real entre el sistema de coordenadas (S) y la parte inferior de la carcasa de plástico está garantizada que esa menos de 3 °.

La ortogonalidad entre los ejes del sistema de coordenadas fijo (S) y la carcasa es menor de 0,1º. Esto también significa que la producción de la aceleración lineal 3D, la tasa de vuelta 3D (giro) y los datos del campo magnético 3D tendrán lecturas ortogonales XYZ dentro de menos de 0,1 °.

- CALIBRADO DEL SISTEMA DE COORDENADAS

El MTi calcula la orientación entre el sistema de coordenadas fijo del sensor (S) y un sistema de coordenadas fijo de referencia a la Tierra (G). Por defecto el sistema de coordenadas de referencia local fijo a la Tierra utilizado se define como un sistema de coordenadas cartesiano con:

- X positivo cuando apunta al norte magnético local.
- Y positivo cuando apunta al Oeste.
- Z positivo cuando apunta hacia arriba.

La salida de la orientación 3D (independiente del modo de salida) se define como la orientación entre el sistema de coordenadas fijo al sensor (S) y el sistema de coordenadas fijo a la Tierra (G) (mostrado en la figura 8), utilizando el sistema de coordenadas fijo a la Tierra (G) como referencia del sistema de coordenadas.

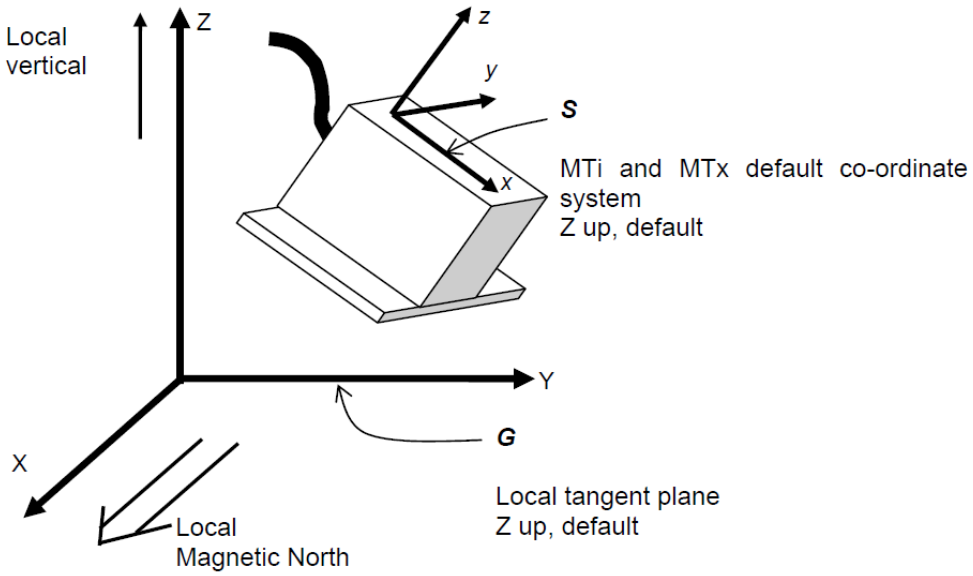

Figura 8: MTi en el sistema de coordenadas de la tierra

La salida de sistema de coordenadas del MTi está referenciado respecto al norte magnético local. La desviación entre el norte magnético y el norte verdadero (conocida como la declinación magnética) varía dependiendo de su ubicación en la tierra y pueden ser obtenidos a partir de unos modelos diferentes del campo magnético de la Tierra como una función de la latitud y longitud. El MTi puede aceptar un ajuste del valor de la declinación. Esto se hace mediante el establecimiento de la declinación en el Administrador de MT, por el kit de desarrollo software (SDK, por sus siglas en inglés) o por comunicación directa con el sensor. Entonces la salida se verá compensado por la declinación y por lo tanto hace referencia al norte verdadero "local".

2.2.3.2 Especificación del rendimiento de orientación

Las características típicas del rendimiento del MTi de la salida de orientación.

Rango dinámico: todos los ángulos en 3D
Resolución angular: 0.05°
Repetitividad: 0.2°

Precisión estática:(roll/pitch): 0.5°

Static Accuracy (heading):1.0°

Precisión dinámica: 2° RMS

Actualización de frecuencia: configurable por el usuario, máximo de 120 Hz

2.2.3.3 Modos de salidas de orientación

La orientación calculada por el MTi es la orientación del sistema de sensores fijos de coordenadas (S) con respecto a un sistema de coordenadas cartesianas fijo a la Tierra (G). La orientación de salida se puede presentar en diferentes parametrizaciones:

- Cuaternios Unidad (también conocidos como parámetros de Euler)

- Ángulos de Euler: balance (roll), cabeceo (pitch), guiñada (yaw)

- Matriz de rotación

2.2.3.4 Modos de salida de datos calibrados

Esta sección está dedicada a dar información detallada sobre la definición de los modos de salida de datos inerciales calibrados del MTi.

FISICA DEL SENSOR

Los sensores físicos dentro del MTi (acelerómetros, giroscopios y magnetómetros) son calibrados de acuerdo a un modelo físico de la respuesta de los sensores a las diferentes magnitudes físicas, por ejemplo, la temperatura. El modelo básico es lineal y de acuerdo con la siguiente relación:

$$\mathbf{s} = K_T^{-1}(\mathbf{u} - \mathbf{b}_\mathbf{T})$$

El modelo utilizado realmente es más complicado y se están desarrollando

constantemente nuevos métodos. A la calibración de cada MTi fabricado se le ha asignado una matriz de ganancia única, y el vector de polarización, . Estos datos de calibración se utilizan para relacionar las tensiones de la muestra digital, **u**, (enteros sin signo de 16 bit salida de un conversor analógico-digital) de los sensores a las respectivas cantidades físicas, **s**.

El aumento de la matriz se divide en una matriz de desalineación, **A**, y una matriz de ganancia, **G**. La desalineación determina la dirección de los ejes con respecto a los ejes del sistema de coordenadas de la carcasa del sensor (S). Por ejemplo, el primer elemento de la matriz de desalineación del acelerómetro $a_{1,x}$ describe la sensibilidad a la dirección del acelerómetro en el canal uno. Las tres direcciones se utilizan para formar la alineación de la matriz:

$$A = \begin{bmatrix} a_{1,x} & a_{1,y} & a_{1,z} \\ a_{2,x} & a_{2,y} & a_{2,z} \\ a_{3,x} & a_{3,y} & a_{3,z} \end{bmatrix} \qquad G = \begin{bmatrix} G_1 & 0 & 0 \\ 0 & G_2 & 0 \\ 0 & 0 & G_3 \end{bmatrix}$$

$$K_T = \begin{bmatrix} G_1 & 0 & 0 \\ 0 & G_2 & 0 \\ 0 & 0 & G_3 \end{bmatrix} \begin{bmatrix} a_{1,x} & a_{1,y} & a_{1,z} \\ a_{2,x} & a_{2,y} & a_{2,z} \\ a_{3,x} & a_{3,y} & a_{3,z} \end{bmatrix} + \mathbf{O}$$

La **O** representa los modelos de mayor orden y modelos de temperatura, correcciones de sensibilidad gravedad, etc.

Cada MTi sigue el modelo de dependencia de la temperatura tanto para ganancia como para sesgo para todos los sensores y otros efectos. Este modelo no está representado en el modelo simple de las ecuaciones anteriores, pero está implementado en el firmware.

MODO DE SALIDA DE DATOS MAGNETICOS Y CALIBRADO INERCIAL

La producción del calibrado de aceleración lineal 3D, el índice de la vuelta 3D (giro) y datos del campo magnético 3D están en el sistema de coordenadas fijo del sensor (S).

Las unidades de la salida calibrada de datos son las mostradas en la tabla 1.

Tabla 3: Unidades de la salida calibrada de datos

Vector	Unit
Acceleration	m/s^2
Angular velocity (rate of turn)	rad/s
Magnetic field	a.u. (arbitrary units) normalized to earth field strength

Los datos calibrados están "sin procesar", es decir, sólo el modelo de calibración física se aplica a los valores de 16-bit encontrado en los convertidores analógico-digital. No hay un filtrado adicional, u otro proceso temporal aplicado al dato. La composición de MTData es el mostrado en la figura 9.

The output definition in calibrated data output mode is:

MTData DATA =

accX	accY	accZ	gyrX	gyrY	gyrZ	magX	magY	magZ

MID 50 (0x32)

All data elements in DATA field are FLOATS (4 bytes) , unless specified otherwise by modifying the OutputSetting Data Format field.

Figura 9: Composición de MTData con calibrado

El acelerómetro / velocidad de giro / datos del magnetómetro pueden ser habilitado o deshabilitado por la función SetOutputSettings.

MODO DE SALIDA SIN CALIBRAR

En el formato de salida no calibrada pura, significa que el modelo de calibración físicas descritas en el apartado anterior no se aplica. Esto le da libre acceso al nivel básico de la unidad del sensor, pero en la mayoría de los casos este nivel de uso no se recomienda. Sin embargo, si su objetivo principal es para el registro y el procesamiento posterior, puede ser ventajoso ya que siempre es posible volver a la "fuente" de la señal. En este modo la temperatura del dispositivo también se envía como se muestra en la figura 10.

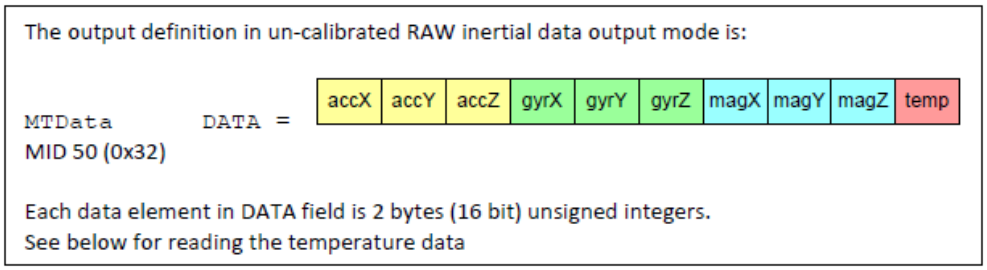

Figura 10: Componentes de MTData sin calibrar

Los dos bytes de datos de temperatura en el modo de salida no calibradas del MTi se puede interpretar como 16 bits con complemento a 2. Sin embargo, debe tenerse presente que la resolución del sensor de temperatura no es en realidad de 16 bits, sino de 12 bits.

2.2.3.5 Restablecer el sistema de coordenadas de referencia

En algunas situaciones puede ocurrir que los ejes del sensor MT no están exactamente alineados con los ejes del objeto por lo que la orientación tiene que ser grabada. Este método permitirá, a la salida de datos de orientación y / o calibrado de inercia en un marco de objeto fijo. El software dispone de cuatro métodos para facilitar la obtención de la salida deseada respecto al sistema de coordenadas, que se muestran a continuación.

Método 1) Configuración de una matriz de rotación arbitraria para rotar S hacia el objeto elegido como sistema de coordenadas O como se muestra en la figura 11.

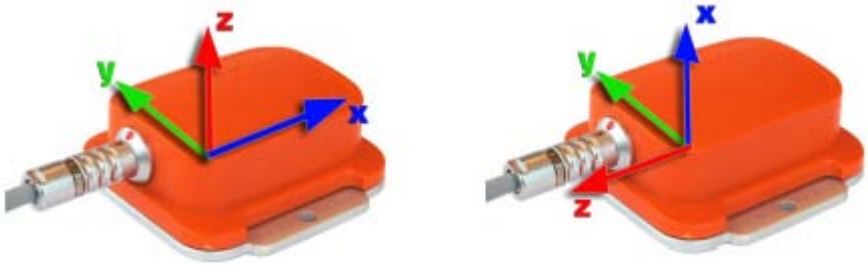

Figura 11: Cambio del sistema de coordenadas con el método 1

Método 2) Es un restablecimiento que redefine el eje X del sistema de coordenadas, manteniendo el eje Z a lo largo de la vertical. Después de restablecer el eje X, la orientación expresada será con respecto a la nueva referencia.

Método 3) Un restablecimiento objeto se define cómo el sensor está orientado con respecto a los ejes de coordenadas a las que se adjunta. Tras el restablecimiento de objetos, tanto la orientación y los datos del sensor calibrado se expresan con respecto a los ejes del objeto. A continuación en la figura 12 se muestra un ejemplo del método 3.

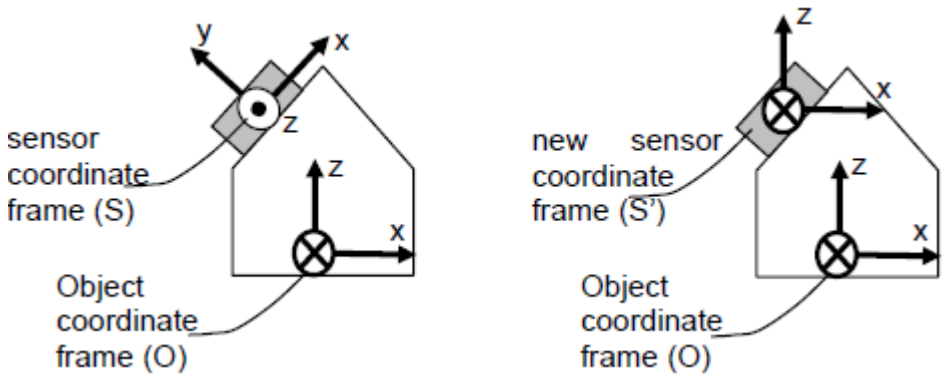

Figura 12: Cambio del sistema de coordenadas con el método 3

Método 4) Es una combinación entre los dos últimos métodos. Este tiene la ventaja de que todos los sistemas de coordenadas se pueden alinear con una sola acción.

2.2.4 Estructura software

En esta sección se describe el concepto de las interfaces CMT en cada nivel donde en la figura 13 se muestra un diagrama de los distintos niveles.

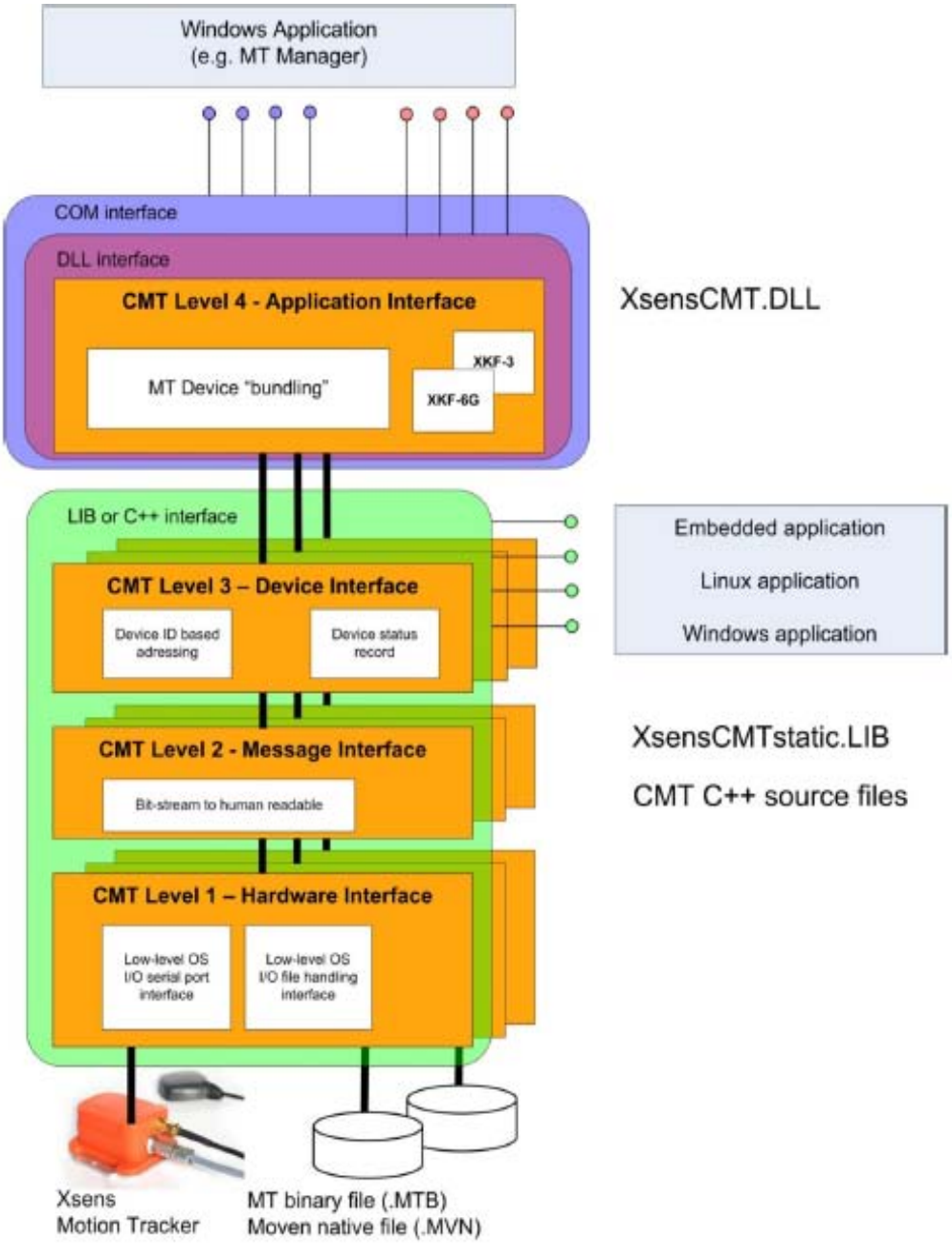

Figura 13: Niveles de CMT

CMT Nivel 1 - Interfaz hardware

Las funciones de este nivel da acceso a las comunicaciones de E / S de bajo nivel. Estas funciones son la plataforma (sistema operativo y el procesador) dependiente de lo que puede tener que ser modificado para aplicaciones empotradas. El propósito de separar a este nivel de los otros niveles CMT es para los programadores de dispositivos integrados para poder facilitar las modificaciones del CMT Nivel 1 código y recompilar la biblioteca hasta CMT Nivel 3 y que puedan utilizar la funcionalidad en alto nivel inmediatamente.

CMT Nivel 2 – Interfaz de mensaje

La interfaz mensaje CMT Nivel 2 reúne el mensaje de protocolo comunicación MT usando las funciones L1 que retorna datos de mensaje en crudo. Si la suma de comprobación es correcta, el mensaje completo se transmite al siguiente nivel CMT. CMT Nivel 2 tiene su propio tiempo de espera (s) que son independientes del nivel 1, excepto que se establezcan disposiciones que entonces si serán más largos que el Nivel 1.

La CMT Nivel 2 interfaz no utiliza ningún otro método específico de direccionamiento más que introduce y utiliza el protocolo de comunicación MT.

Normalmente, no hay necesidad de utilizar directamente la interfaz de la CMT o modificar el nivel 2.

CMT Nivel 3 – Interfaz de dispositivo

La CMT Nivel 3 implementa las siguientes capacidades:

• Mantiene el registro de las configuraciones de dispositivos MT para un solo dispositivo Master MT (puerto de comunicaciones o archivo)

• Este nivel implementa todos los mensajes MT. Por ejemplo gotoConfig, setDeviceMode, getProductCode, etc.

• MT de datos se recupera en una especial estructura de paquetes que interpreta y traduce el contenido de los datos para acceso de datos fácil.

• Este nivel también tiene función de registro, que le permite registrar todos los mensajes recibidos de forma automática y luego leer el archivo.
• Los dispositivos ya están contemplados en deviceid y no por número de puerto serie.

CMT Nivel 3 está disponible para los desarrolladores como el componente más alto nivel de la biblioteca estática y como código fuente.

CMT Nivel 4 - Interfaz de aplicación

El Nivel 4 CMT implementa una gran cantidad de funcionalidad adicional para facilitar su uso y conexión a dispositivos de MT múltiples simultáneamente. Proporciona un contenedor para todos los niveles inferiores y las exportaciones de la CMT Nivel 3 y funcionalidad. Nivel 4 CMT implementa manejo automático de múltiples dispositivos conectados y automático de datos de cola (buffering). Básicamente, la CMT Nivel 4 alivia muchos de los detalles de implementación de la manipulación de MTi del programador de aplicaciones. Nivel 4 permite el registro automático a un archivo, la recepción de múltiples paquetes de datos de MT, el inicio sincronizado automático (software), etc.

El manejo de los mensajes entrantes MT protocolo de comunicación y el solicitar al dispositivo se realiza en un subproceso independiente y la mayoría de los mensajes modo de medición pueden ser enviados y recibidos sin preocuparse de molestar a los datos entrantes.

CMT Nivel 4 está disponible para los desarrolladores como una biblioteca de vínculos dinámicos de Windows (DLL) y como un objeto COM y requiere un entorno de múltiples subprocesos para funcionar. El binario es el mismo para ambas interfaces.

CMT Nivel 4 también apoya la conversión basada en software de datos mediante la aplicación de filtros Kalman Xsens (XKF) para emular el dispositivo de salida directa:

- Los datos en bruto a datos de calibrado
- Los datos calibrados a datos de la orientación (sólo cuando se convirtió originalmente a partir de datos en bruto)
- Los datos calibrados y los datos GPS PVT a la orientación y datos de posición (sólo cuando se convirtió originalmente a partir de datos en bruto)
- Conversión del modo de orientación de un cuaternión a otro (a Euler)

En particular, esto puede ser útil para el procesamiento posterior de los datos registrados binario utilizando diferentes entornos (por ejemplo, Escenarios XKF) o transformación no ocasional.

2.2.5 Especificaciones físicas

En este apartado hablaremos de las especificaciones físicas del sensor como los sensores de los que consta el MTi, las medidas, etc.

2.2.5.1 Visión general de la física del sensor

Tabla 4: Sensores en el MTi

MTi and MTx Sensor Fact Table	
Accelerometers	MEMS solid state, capacitative readout
Rate of turn sensor (rate gyroscope)	MEMS solid state, monolithic, beam structure, capacitative readout
Magnetometer	Thin film magnetoresistive

Además, de estos sensores el MTi tiene varios sensores de temperatura a bordo para permitir la compensación por dependencia de la temperatura de los diferentes sensores.

2.2.5.2 Propiedades físicas generales

VISIÓN GENERAL DEL MTi

Tabla 5: Propiedades físicas del sensor

	MTi-28A##G##
Communication interface:	Serial digital (RS-232)
Additional interfaces:	SyncIn SyncOut Analog In
Operating voltage[22]:	4.5-30 V
Power consumption[23]: (AHRS/3D orientation mode)	350 mW
Temperature Operating Range:	-20°C - 55°C
Specified performance Operating Range:	0°C - 55°C
Outline Dimensions:	58 x 58 x 22 mm (W x L x H)
Weight:	50 g

FUENTE DE ALIMENTACIÓN

Propiedades de la fuente de alimentación:

- La fuente de alimentación nominal del MTi es de 5V DC.

- La tensión mínima de funcionamiento de suministro es > 4,5 y el máximo absoluto es inferior a 30V.

- El sensor trabaja en una fuente de alimentación de> 4.5-30V. Utilice sólo SELV (Separado o de seguridad mediante baja tensión) fuentes de alimentación (doble aislamiento) que son a prueba de cortocircuitos.

- El consumo de energía promedio de operación es 350MW (~ 70 mA a 5 V) para el MTi. El consumo medio de energía puede variar ligeramente con el modo de uso (carga DSP). Tenga en cuenta que la eficiencia de la etapa de entrada de energía se reducirá con el aumento de la tensión de alimentación. A 5 - 6 V de CC, la eficacia es óptima, a 30V de CC, la eficiencia es de alrededor de 75%.

- El pico de corriente en el arranque (encendido) pueden ser de hasta 200mA.

Cuando se opera a temperatura ambiente la temperatura dentro del sensor será 33-40 ° C en condiciones normales.

ESPECIFICACIONES DE LA INTERFAZ FISICA

En la figura 14 se puede observar una vista general del cable USB serie y cable de alimentación.

Figura 14: Cable del sensor MTi

Los datos de serie USB y el cable de alimentación suministrada con el Kit de Desarrollo MTi son compatible con USB 1.1 y superior. Hay que asegurarse de que la salida USB del PC está cualificada para entregar 100 mA o más.

El USB de datos y el cable de alimentación proporcionan un fácil acceso a las clavijas individuales del rastreador de movimiento. Dentro de la carcasa hay un conector libre que puede ser usado por ejemplo para propósitos de sincronización. La figura 15 muestra la ubicación del conector.

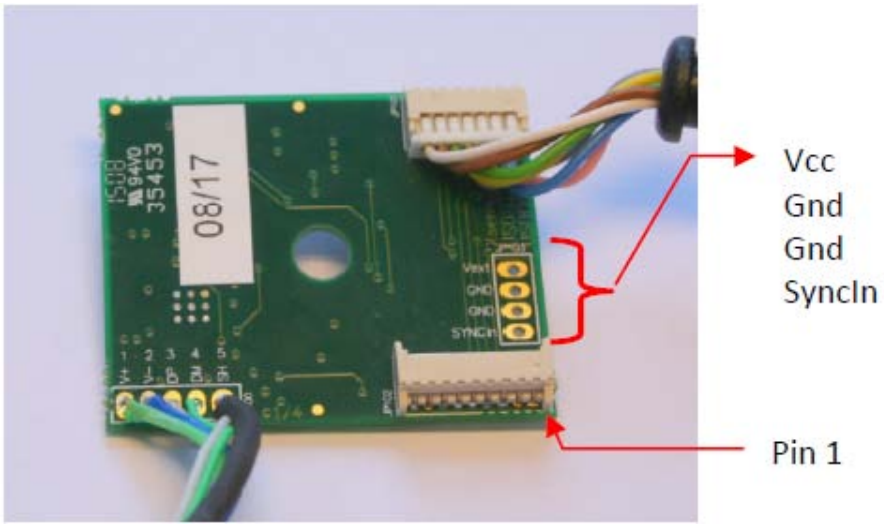

Figura 15: Interior del sensor

Las definiciones de los 7 pines son las que se muestran en la tabla 6.

Tabla 6: Pines del MTi RS-232

Molex pin	MTi RS-232
Pin 1	VCC
Pin 2	GND
Pin 3	Analog IN
Pin 4	TX (sensor)
Pin 5	RX (sensor)
Pin 6	SyncOut
Pin 7	SyncIn

La temperatura de funcionamiento de los datos de serie USB y cable de alimentación (CA-USB) es de 0 ° C - 40 ° C.

El MTi está diseñado para ser utilizado con la alimentación proporcionada por Xsens (integrado en el RS-232 a USB cable). Es posible utilizar otras fuentes de alimentación, sin embargo esto debe hacerse con cuidado. Por razones de seguridad cualquier fuente de alimentación que se utilice con el dispositivo debe cumplir con la directiva de compatibilidad electromagnética.

Los pines y definiciones de los colores del cable del MTi son distintos dependiendo del MTi que se utilice, en nuestro caso MTi-28A # # # # G (MTi RS-232, versión estándar).

El conector hembra del MTi es un ODU serie L de 7 pines como se puede ver en la figura 16 que es una vista donde se ve la numeración de los pines.

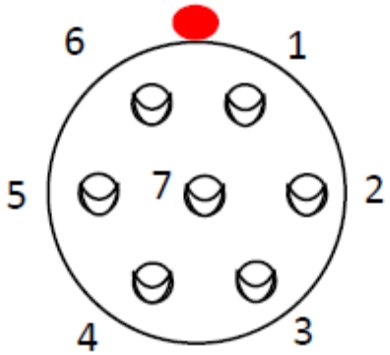

Figura 16: Clavija hembra del sensor

Conector macho del MTi es un ODU serie L de 7 pines que viene del convertidor RS232-USB. A continuación en la figura 17 se muestra la clavija y la numeración de los pines.

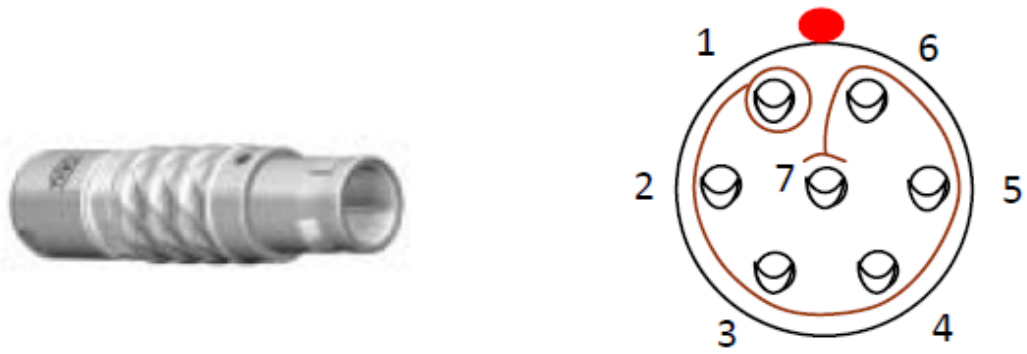

Figura 17: Clavija macho del cable del sensor

La definición de los pines del conector del MTi se muestran en la tabla 7 y el color del cable se muestra en la tabla 8:

Tabla 7: Pines del conector ODU

Signal	ODU pin
VCC	Pin 1
GND	Pin 2
Y / A	Pin 3
Z / B	Pin 4
Reserved	Pin 5
SyncOut	Pin 6
SyncIn	Pin 7

Tabla 8: Colores de los cables

ODU pin	Unitronic cable	Elitronic cable
Pin 1	Yellow	White
Pin 2	Yellow-green	Brown
Pin 3	Black	Green
Pin 4	Beige	Yellow
Pin 5	Brown	Grey
Pin 6	Green	Pink
Pin 7	Blue	Blue

ESPECIFICACIONES MECÁNICAS DE LA CARCASA

Las piezas de plástico de la carcasa son de poliamida (PA6.6). La placa inferior MTi está hecha de aluminio anodizado (6082). La carcasa es a prueba de polvo, pero no a prueba de agua. La toma de conector de MTi y ensamblaje de la carcasa de goma en forma de anillo y es generalmente más robusta para entornos difíciles.

PROTECCIÓN DE LA CARCASA DEL MEDIO AMBIENTE

La carcasa MTi está diseñada para soportar el uso en aplicaciones donde puede estar protegido del polvo y salpicaduras de agua en ciertas ocasionales. Los test de Xsens han confirmado que la carcasa y el conector puede soportar circunstancias ambientales temporales equivalentes a las de protección IP 66 (selladas contra el polvo, protección contra el chorro de agua de gran alcance). Tenga en cuenta que el conector de la carcasa MTi es resistente al agua, pero el conector que se suministra no es a prueba de agua.

El material plástico utilizado para MTi tiene la clasificación UL94 V2.

Las dimensiones del MTi se muestran a continuación en la figura 18.

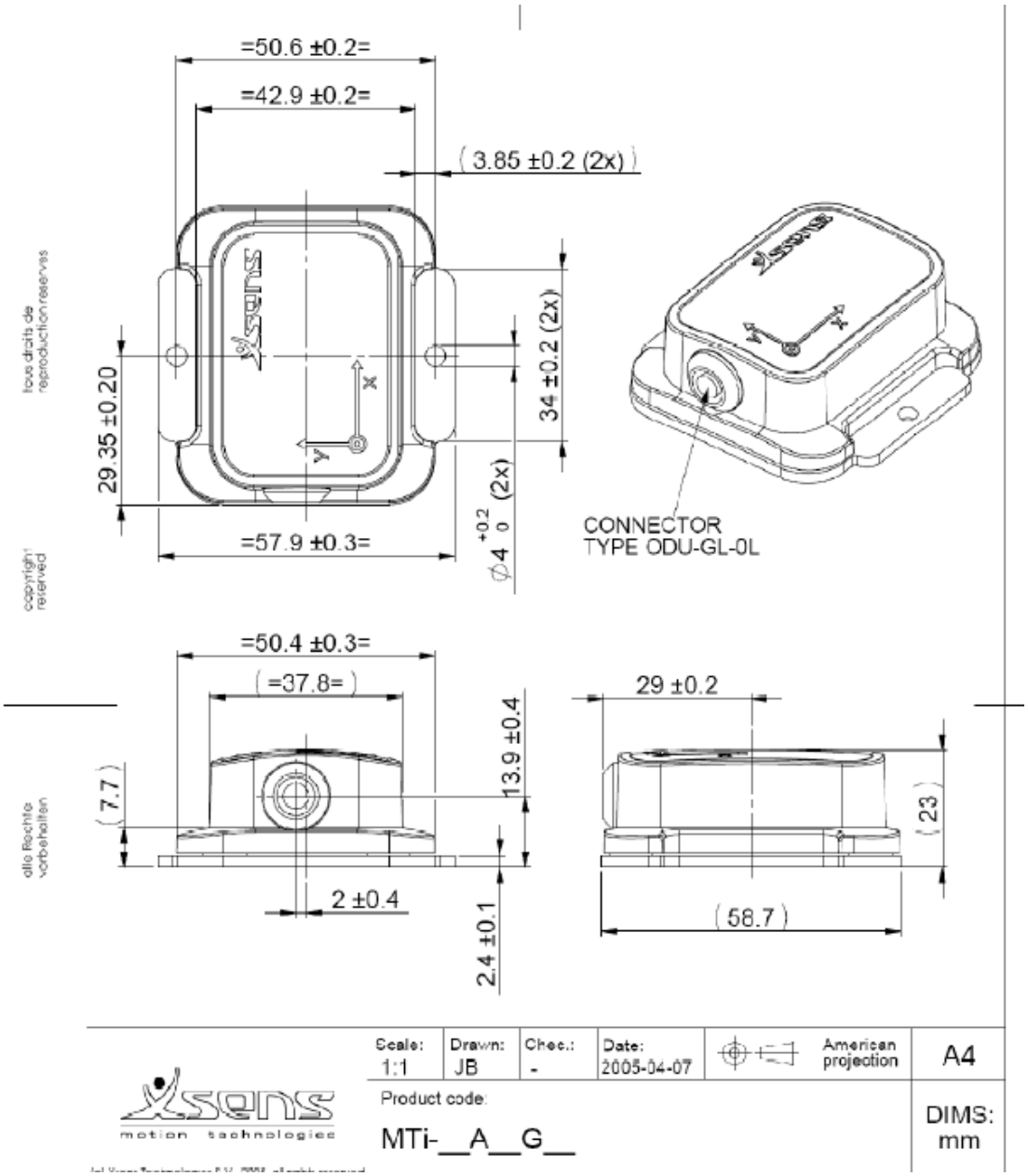

Figura 18: Medidas del sensor

LOCALIZACIÓN DEL ORIGEN FÍSICO

El MTi es ante todo un sensor de orientación y como tal no es importante el que tiene su origen interno, es decir, la orientación es la misma para todas las posiciones de la MT, ya que puede ser considerado como un cuerpo rígido. Sin embargo, para aplicaciones

donde se miden las aceleraciones que es importante conocer el verdadero origen de la MT, se define por la ubicación física del acelerómetro.

A continuación como se puede observar en la figura 19 se muestra cómo puede encontrar el vector de traslación entre el origen O de la MT y un cierto punto conveniente dispositivo externo O '(un agujero del tornillo) u O" (la intersección entre las partes del MTi) en el exterior de la carcasa.

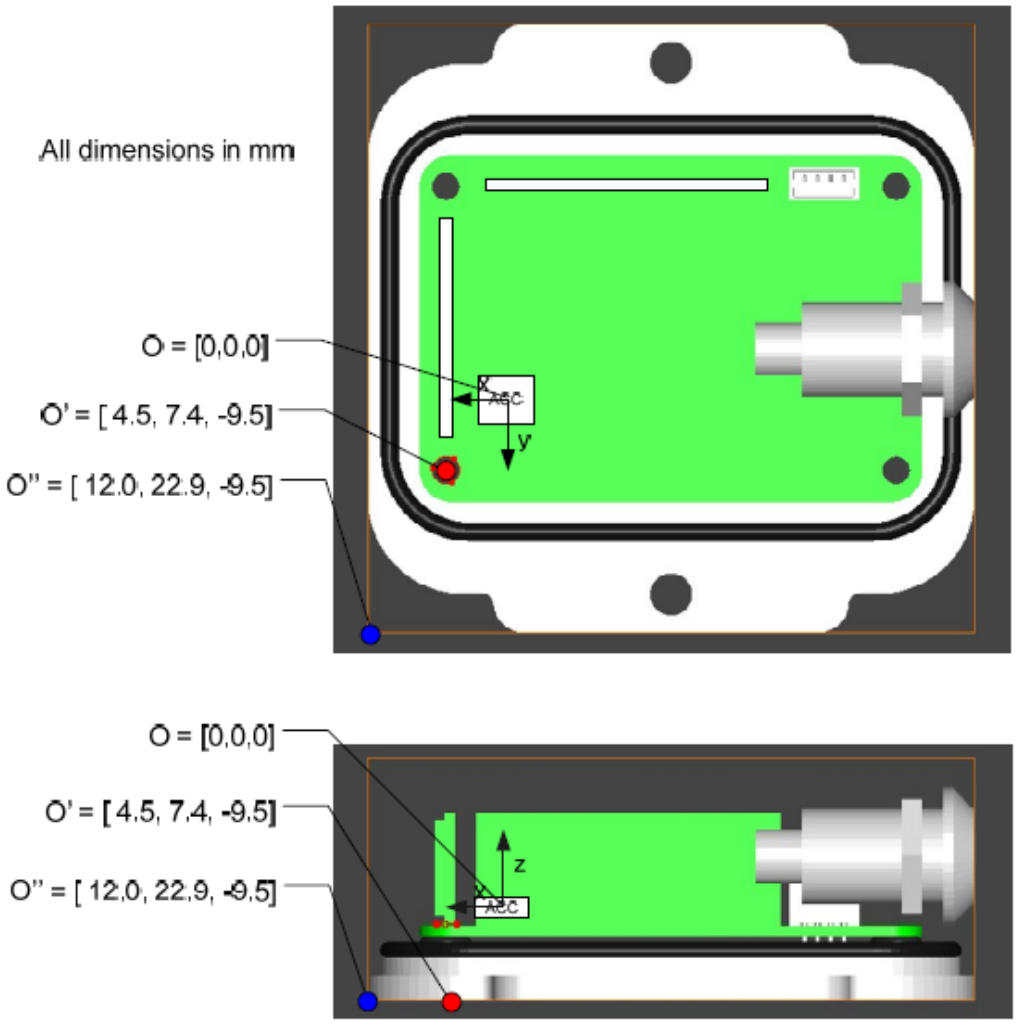

Figura 19: Posición del eje de coordenadas

CONSIDERACIONES

Hay tres consideraciones que hay que tener en cuenta a la hora de colocar el MTi en un objeto o cuerpo y son las aceleraciones transitorias, las vibraciones y los materiales magnéticos.

Aceleraciones transitorias

Los acelerómetros lineales 3D en el MTi se utilizan principalmente para estimar la dirección de la gravedad para obtener una referencia del sensor (pitch / roll). Durante largos períodos de tiempo (más de unos pocos segundos) de aceleraciones transitorias (es decir, derivado de la segunda posición) la observación de la gravedad no se puede hacer. Los algoritmos de fusión de sensores XKF toman en cuenta estos efectos, pero sin embargo es imposible estimar la verdadera vertical sin información adicional.

El impacto de aceleraciones transitorias se puede minimizar si se toma en cuenta algunas consideraciones durante la colocación del dispositivo.

Si desea utilizar el MTi para medir la dinámica de un vehículo en movimiento, embarcaciones, lo mejor es colocar el dispositivo de medición en una posición donde se espera que hayan menos aceleraciones transitorias. Esto es típicamente cerca del centro de gravedad del vehículo / embarcación, ya que cualquier rotación alrededor del centro de gravedad se traduce en aceleraciones centrípetas en cualquier punto fuera del punto de rotación, que es generalmente cerca del centro de gravedad. La aceleración del vehículo en su conjunto no puede tenerse en cuenta.

Vibraciones

Para un mejor funcionamiento del MTi, deberá estar mecánicamente aislado de las vibraciones tanto como sea posible. Las vibraciones se miden directamente por los acelerómetros. Esto no es necesariamente un

problema, pero dos condiciones pueden hacer las lecturas de los acelerómetros inválidas:

1) La magnitud de la vibración es mayor que el rango del acelerómetro. Esto hará que el acelerómetro se sature, lo que puede ser observado como una "deriva" en el nivel cero del acelerómetro. El mismo se mostrará en las estimaciones de la orientación 3D como un error de roll/pitch.

2) La frecuencia de la vibración es mayor que el ancho de banda del acelerómetro. En teoría, estas vibraciones son rechazadas, pero en la práctica todavía puede dar lugar a aliasing, sobre todo si está cerca del límite de ancho de banda. Esto se puede observar como una oscilación de baja frecuencia. Además, vibraciones de alta frecuencia a menudo tienden a tener grandes amplitudes de aceleración.

Los materiales magnéticos y los imanes

Cuando un MTi se coloca cerca o sobre un objeto que contiene los materiales ferromagnéticos, o que es magnético por sí mismo, la medida del campo magnético es distorsionada y provoca un error en la medida de orientación. El campo magnético terrestre se ve alterado por los materiales ferromagnéticos, los imanes permanentes o corrientes muy fuertes (varios amperios). En la práctica, la distancia al objeto y la cantidad de material ferromagnético determina el grado de perturbación. Los errores en la orientación debido a estas distorsiones pueden ser muy grandes, ya que el campo magnético terrestre es muy débil en comparación con la magnitud de las muchas fuentes de distorsión.

Sea o no un objeto ferromagnético preferentemente deberá ser revisado por el uso de los magnetómetros del MTi. También se puede comprobar con un pequeño imán, pero tenga cuidado, usted puede fácilmente duro magnetizar materiales ferromagnéticos, provocando errores aún más

grandes. Si usted encuentra que algún objeto se encuentra magnetizado puede ser posible "la desmagnetización" del objeto.

En la mayoría de los casos en que la alteración del campo magnético causado por la colocación del MTi en un objeto ferromagnético se puede corregir para el uso de un procedimiento de calibración. El procedimiento de calibración se puede ejecutar en pocos minutos y produce un nuevo conjunto de parámetros de calibración que se puede escribir en la memoria no volatil del MTi.

Este procedimiento de calibración se lleva a cabo en el módulo de software "Campo Magnético Mapper" que viene con el SDK. El método utilizado en este software es único en el sentido de que permite a un usuario elegido secuencia de medición (con ciertas limitaciones), y que permite el mapeo 3D.

2.3 Robots Component Guidelines v0.3

El módulo que se ha diseñado está englobado dentro del proyecto ASIBOT desarrollado por el grupo de investigación Robotics Lab del Departamento de Ingeniería de Sistemas y Automática de la UC3M.

Para conseguir crear un entorno de trabajo cómodo y accesible por parte de todos los alumnos, investigadores y desarrolladores, existen una relación de guías utilizadas que es importante respetar (RCGv03) [6].

Toda esta información se encuentra perfectamente documentada en la Wiki de robots de la UC3M [7], dentro del apartado ASIBOT.

Siguiendo las recomendaciones que aquí se encuentran, se procedió a instalar los paquetes necesarios para la adaptación del módulo propio.

RCGv03 tiene 4 aspectos fundamentales que son: formato de comandos, estructura de carpetas, versión de librerías y nombramiento de módulos.

Estas directivas son necesarias para compatibilizar los distintos módulos que se han desarrollado y que se desarrollarán en un futuro, siendo necesario incluir en el repositorio las novedades que se utilicen por primera vez en un módulo.

Dicho repositorio de Robotics Lab perteneciente al desarrollo del ASIBOT está basado en Subversion, que es una herramienta software para la gestión de ficheros online y se puede encontrar un tutorial de uso en español en la dirección:

http://asrob.uc3m.es/w/index.php/Tutorial_SVN

En primer lugar es importante comprobar las dependencias que se van a tener respecto a las librerías y comprobar si se tiene instalada en el ordenador la versión indicada.

A continuación es recomendable echar un vistazo a los nombres utilizados para definir las variables y si es posible utilizar las disponibles en el repositorio.

El siguiente paso es crear el proyecto en cuestión en la rama del repositorio en la cual estemos trabajando y se puede empezar a generar código, pero respetando el formato de comando que corresponda a la categoría del módulo propio.

A continuación se detallan las directivas de las que se ha hecho referencia anteriormente y las utilizadas en este módulo.

- COMMAND FORMAT
  El formato de comandos del módulo propio es el correspondiente a la de la categoría “Module: sns_name” esta denominación es debido a que es un sensor.

- FOLDER STRUCTURE
  Se recomienda la siguiente estructura de carpetas de proyecto:

- **AUTHORS**
- CmakeFiles.txt
- config
- doc
  - Doxyfile
- INSTALL
- out
  - share
  - -win32
- src
  - .cpp
  - .h

Las cuales se irán completando a medida que se desarrolla el módulo.

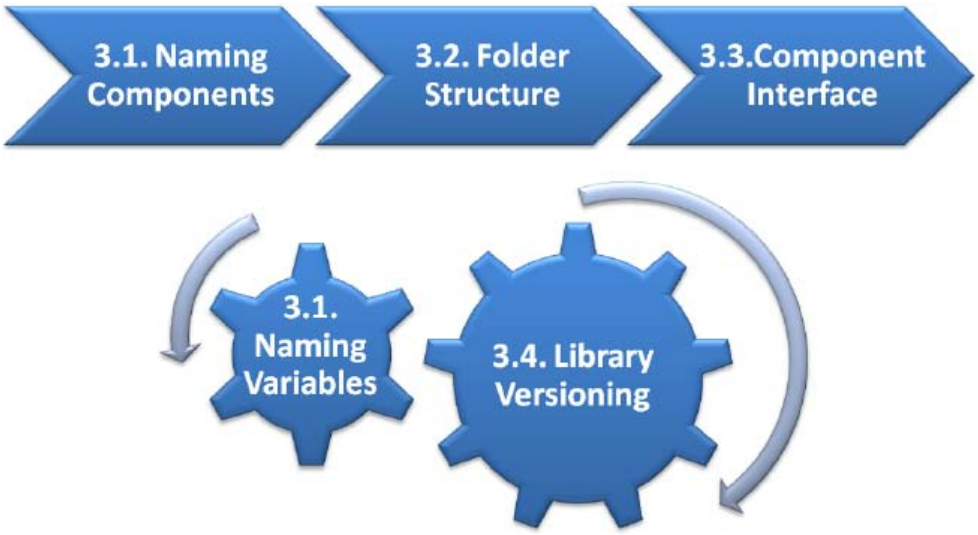

Figura 20: RCGv03. Metodología para creación de componentes

- NAMING MODULES

  Las directrices para nombrar el módulo son que las tres primeras letras indican la naturaleza del mismo. En este caso “sns_”. A continuación se pone una descripción del mismo en nuestro caso es la marca del sensor con esto el modulo se llama “sns_xsens”.

  Las variables que relaciona algún aspecto comunicativo tiene que enviar un pequeño nombre que identifique el tipo de dato y a continuación el dato.

- LIBRARY VERSIONING

  En el repositorio se encuentran la mayoría de las librerías necesarias para el correcto funcionamiento de los programas a desarrollar, siendo recomendable incluir cualquier librería usada por primera vez en dicho repositorio. Las librerías a incluir se detallan a continuación en los siguientes puntos.

2.3.1 Arquitectura software YARP

YARP es una arquitectura software, es decir, un conjunto de bibliotecas, protocolos y herramientas para mantener los módulos y dispositivos disociados para hacer que el software robot sea más estable y de larga duración, sin poner en peligro la capacidad para cambiar constantemente los sensores, actuadores, procesadores y redes [8]. Además, ayuda a organizar la comunicación entre los sensores, procesadores y actuadores para que el acoplamiento sea flexible, para que la evolución del sistema sea progresiva y mucho más fácil.

El modelo de comunicación YARP es de transporte neutral, de modo que el flujo de datos está desconectado de los detalles de las redes y protocolos en uso (permitiendo usar distintos de forma simultánea). YARP utiliza una metodología para la interfaz con los dispositivos (sensores, actuadores, etc.) que hace hincapié en la posibilidad de articulación flexible y hacer cambios en los dispositivos menos perjudiciales. Al mismo tiempo, YARP quiere reducir al mínimo el problema de la incompatibilidad de "arquitecturas", "infraestructuras", y "middleware" (también conocido en este contexto como "muddleware").

YARP es de código abierto y dependiente de la librería ACE (Adaptive Communication Environment). Junto con muchos otros beneficios, el Software Libre puede acelerar el desarrollo de software para las pequeñas comunidades, está escrito por y para investigadores en robótica, en especial la robótica humanoide, que se encuentran con una multitud de hardware complicado de controlar y otra multitud de software igual de

complicado. En definitiva YARP es un conjunto de herramientas de utilidad para satisfacer las necesidades de cómputo para el control de robots humanoides y robots en general.

2.3.1.1 Componentes de YARP

Los componentes de YARP puede dividirse en:

- **libYARP_OS** - interfaz con el sistema operativo (s) para apoyar la fácil transmisión de datos a través de muchos hilos a través de muchas máquinas. YARP está escrito para ser neutral frente a sistemas operativos, y ha sido utilizado en Linux, Microsoft Windows, Mac OSX y Solaris. YARP utiliza la biblioteca ACE de código abierto (Adaptive Communication Environment), que es portable a través de una amplia gama de entornos muy extensa, y hereda esta portabilidad. YARP está escrito casi en su totalidad en C++.

- **libYARP_sig** - realizar las tareas de procesamiento de señal común (visual, auditivo) de una manera abierta fácilmente interconectado con otras bibliotecas utilizan comúnmente, por ejemplo OpenCV.

- **libYARP_dev** - interfaz con dispositivos comunes usados en robótica: cámaras digitales, tarjetas de control de motor, etc.

Estos componentes se mantienen por separado. El componente básico es libYARP_OS, que deberá estar disponible antes de que otros componentes puedan ser utilizados.

En YARP hay tres niveles de configuración: sistema operativo, hardware, y el nivel de robot.

El primer nivel de configuración debe referirse sólo si estás planeando compilar YARP en un sistema operativo nuevo.

El segundo nivel es el hardware. Una nueva adición en una plataforma existente o una plataforma completamente nueva puede requerir la preparación de algunos conductores YARP dispositivo. Estos son a todos los efectos clases C++ que soportan los métodos de acceso al hardware que normalmente se implementa a través de llamadas a función a lo proporcionado por el proveedor de hardware. Esto viene típicamente en la forma de una DLL o una biblioteca estática.

Por último, se pueden preparar los archivos de configuración de una plataforma robótica completamente nueva.

2.3.1.2 Comandos de YARP (versión 2.2.6)

Los comandos que se pueden utilizar son:

help: Sirve para obtener esta lista.
version: obtener información de la versión.
detect: buscar el servidor YARP del nombre que le pongas.
where: informe del servidor YARP que le indiques que se está ejecutando.
conf: Ubicación del informe del archivo de configuración y, opcionalmente, configurarlo.
name : enviar comandos al servidor YARP que le indiques.
connect: crear una conexión entre dos puertos.
disconnect: desconecta una conexión entre dos puertos.
read: lectura de la red y de impresión en la salida estándar.
write: escribir a la red de la entrada estándar.
readwrite: lectura de la netwok e imprimir en la salida estándar, escriba a la red de la entrada estándar.
rpc: lectura / escritura de comandos a un puerto, en el formato estándar.
rpcserver: hacer un servidor de prueba de RPC para recibir y responder a los mensajes en formato de botella.

forward: adelanta comandos a un puerto, en el formato estándar.

regression: realizar pruebas de regresión si vinculados.

server: ejecutar el servidor YARP que le indiques.

check : ejecutar una prueba simple para ver si está trabajando YARP.

terminate: poner fin a un proceso de YARP-terminar con conciencia por su nombre.

ping: obtener información en directo sobre un puerto.

exists: comprobar si un puerto o una conexión está activa.

wait: esperar a que un puerto este activado.

cmake: crear archivos para ayudar a compilar proyectos YARP.

run: iniciar y detener los procesos.

namespace: establecer o consultar el nombre del servidor YARP que tu le indiques (por defecto es /root).

clean: tratar de eliminar las entradas inactivas desde el servidor que le introduzcas.

resource: localiza los archivos de recursos .

yarpview: es un visor de imagen que acepta los datos de entrada por un puerto de YARP.

Figura 21: Yarpview

2.3.2 Cmake

Es una herramienta multiplataforma de generación o automatización de código. El nombre es una abreviatura de "cross platform make" (make multiplataforma).

CMake es una familia de herramientas diseñada para construir, probar y empaquetar software. Se utiliza para controlar el proceso de compilación del software usando ficheros de configuración sencillos e independientes de la plataforma. CMake genera makefiles nativos y espacios de trabajo que pueden usarse en el entorno de desarrollo deseado. Soporta la generación de ficheros para varios sistemas operativos tales como GNU/Linux, Windows, Mac OS/X, etc.

El proceso de construcción se controla creando uno o más ficheros CMakeLists.txt en cada directorio (incluyendo subdirectorios). Cada CMakeLists.txt consiste en una o más directivas. Cada directiva tiene la forma COMANDO (argumentos...) donde COMANDO es el nombre de la directiva, y argumentos es una lista de argumentos

separados por espacios. CMake provee unas directivas predefinidas y otras definidas por el usuario.

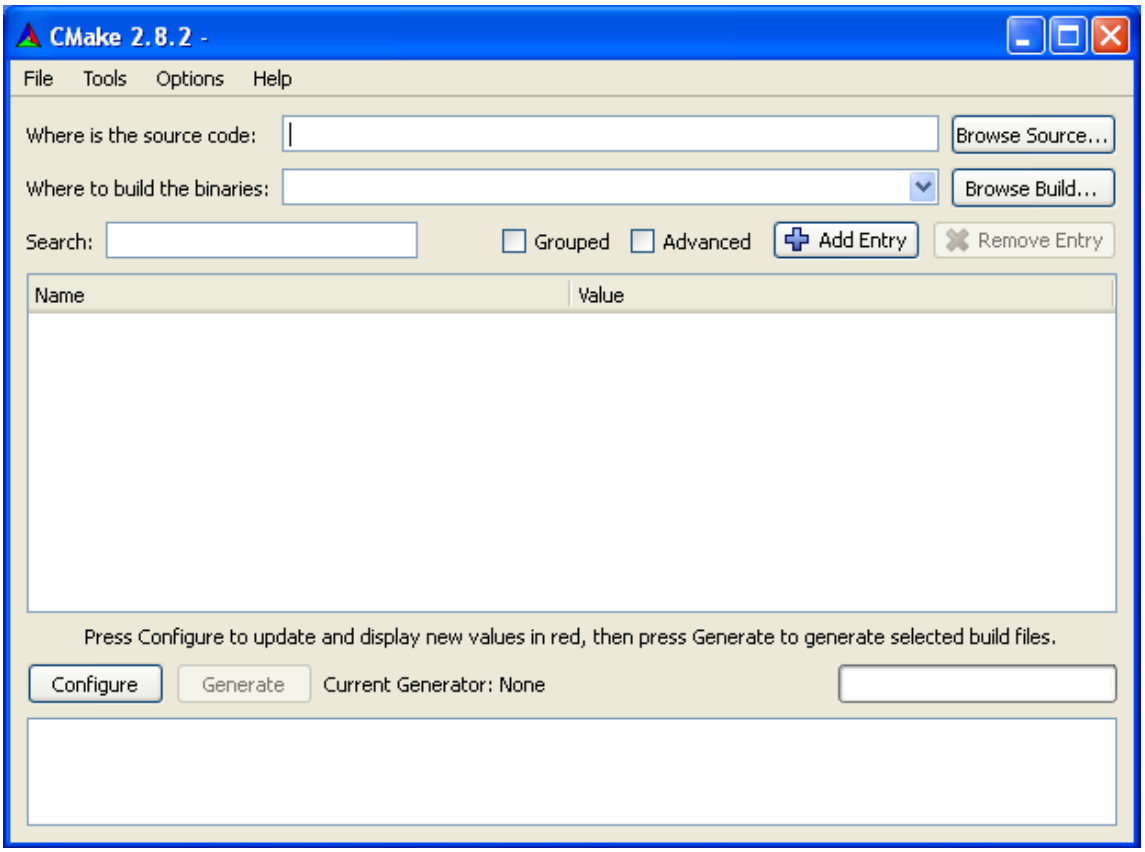

Figura 22: CMake

2.4 Otras herramientas software utilizadas

TortoiseSVN es un cliente gráfico para repositorios basados en Subversion (SVN). Utiliza un servidor centralizado al cual se conectan los diversos usuarios del sistema para subir o descargar ficheros. Se actualiza con las subidas de ficheros de los usuarios, pero las anteriores versiones se mantienen y son siempre accesibles en caso de necesidad de retroceder. Además registra que cambios existen entre distintas revisiones de código.

Figura 23: TortoiseSVN

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Figura 24: Visual Studio

XSENSCMT.DLL es la biblioteca de enlace dinámico que nos proporciona XSENS para poder comunicarnos con el sensor de una manera más sencilla

.

3 Desarrollo software

En este capítulo se hablará de pasos los pasos para poder programar, se explicará el código y además como poner en marcha el componente software.

3.1 Pasos para poder programar

Lo primero para poner en marcha la programación del componente software para robótica es bajarse el Tortoise SVN para poder compartir online las evoluciones del código y poder retroceder si lo necesitáramos.

Lo siguiente es instalar el Visual Studio y una vez que se instale Visual Studio se pasara a instalar la librería ACE que es necesario para instalar YARP para ello una vez bajado ACE se busca en su carpeta y se renombra “config-win32.h” por el nombre “ config.h”. Después se busca "ACE_wrappers_vc9.sln" y se abre con Visual Studio. Tras abrir el proyecto se cambia el tipo de construcción de Debug a Release como se ve en la figura 25 y se presiona Build --> Build Solution.

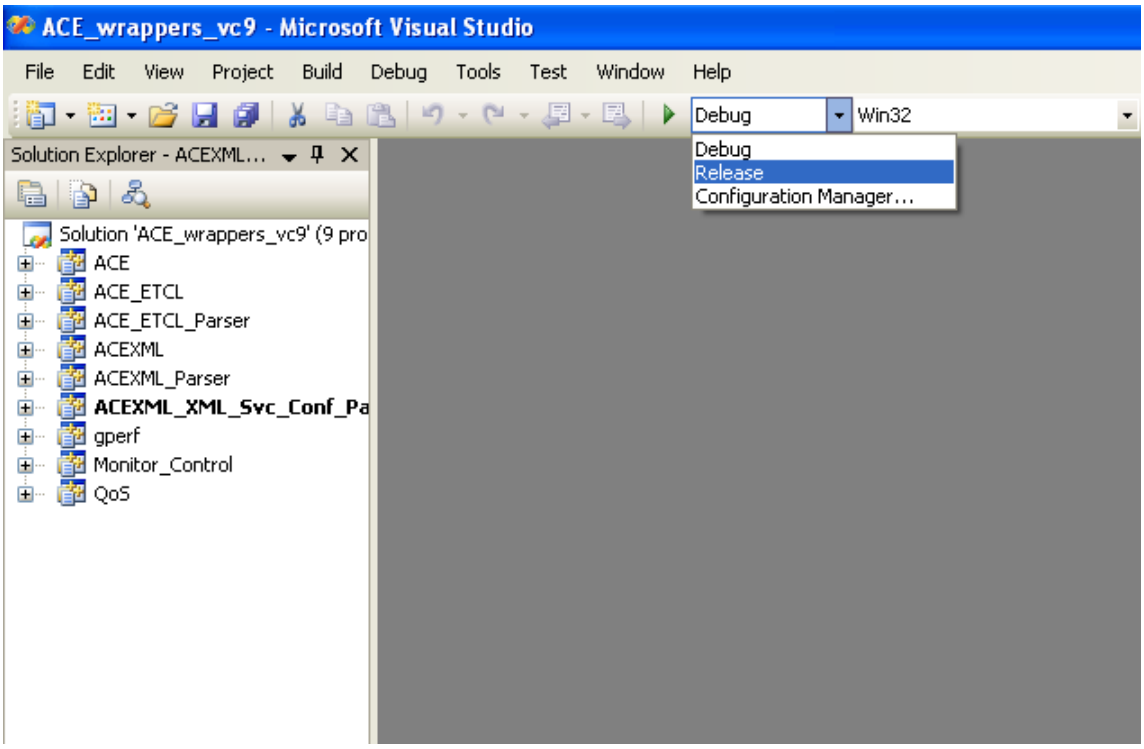

Figura 25: Instalación de ACE

Para instalar YARP 2.2.6, primero se guardar el archivo en una carpeta determinada. Después se crea una carpeta llamada build dentro de la carpeta de YARP. Luego se abre CMake. Tras abrir CMake en donde pone “where is the source code ” se introduce la ruta de donde está la carpeta YARP y en donde pone “where to build the binaries” se pone la ruta de la carpeta build que se ha creado anteriormente. Después de introducir la dirección de las carpetas se pulsa Configuración, selecciona VS2008 y se presiona finalizar. Posteriormente se pulsa a configurar de nuevo y después a generar (si da algún error es debido a que no encuentra algún archivo y se tendrá que ponerse manualmente).

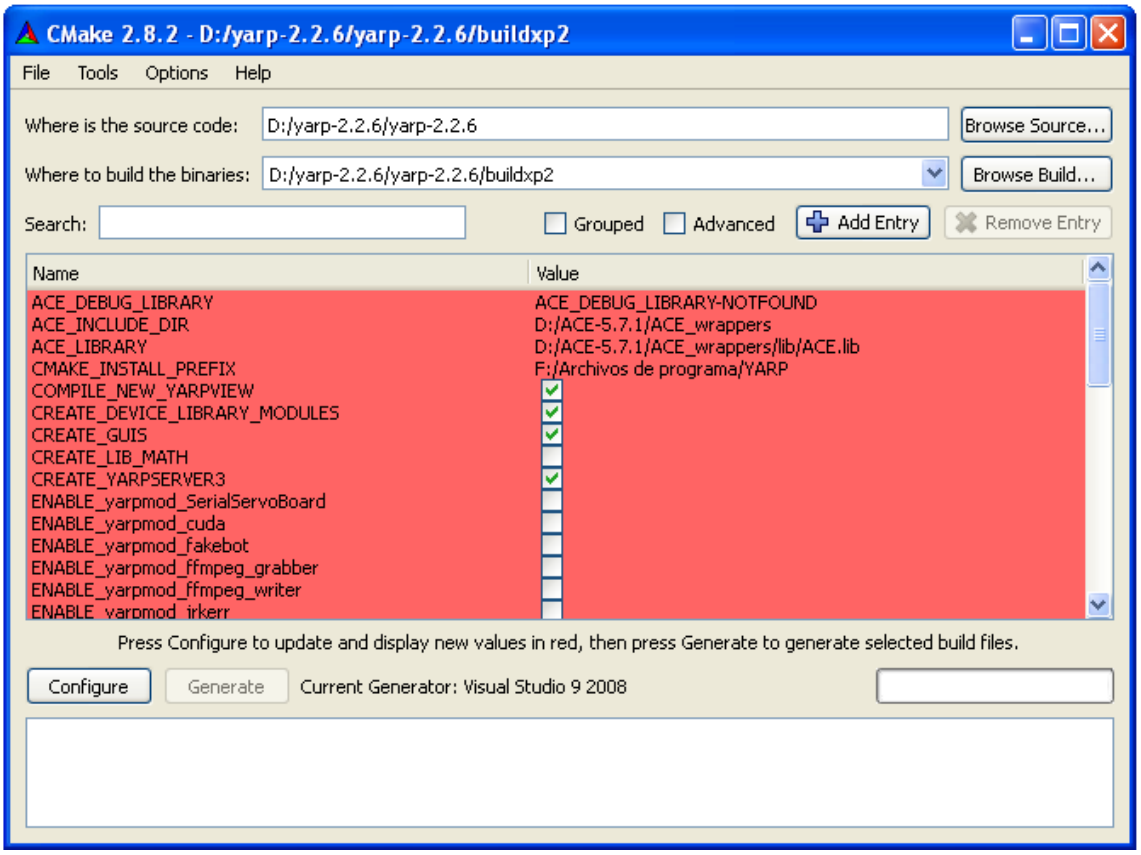

Figura 26: CMake creando el proyecto para compilar YARP

Se cierra CMake y se va a la carpeta build que se ha creado en ella el proyecto “YARP.sln”. Se abre y después cambias el tipo de construcción de Debug a Release y se presiona Build --> Build Solution. Tras construir la solución se busca en el “solution explorer” instal y se pulsa el botón derecho del ratón y se presiona build.

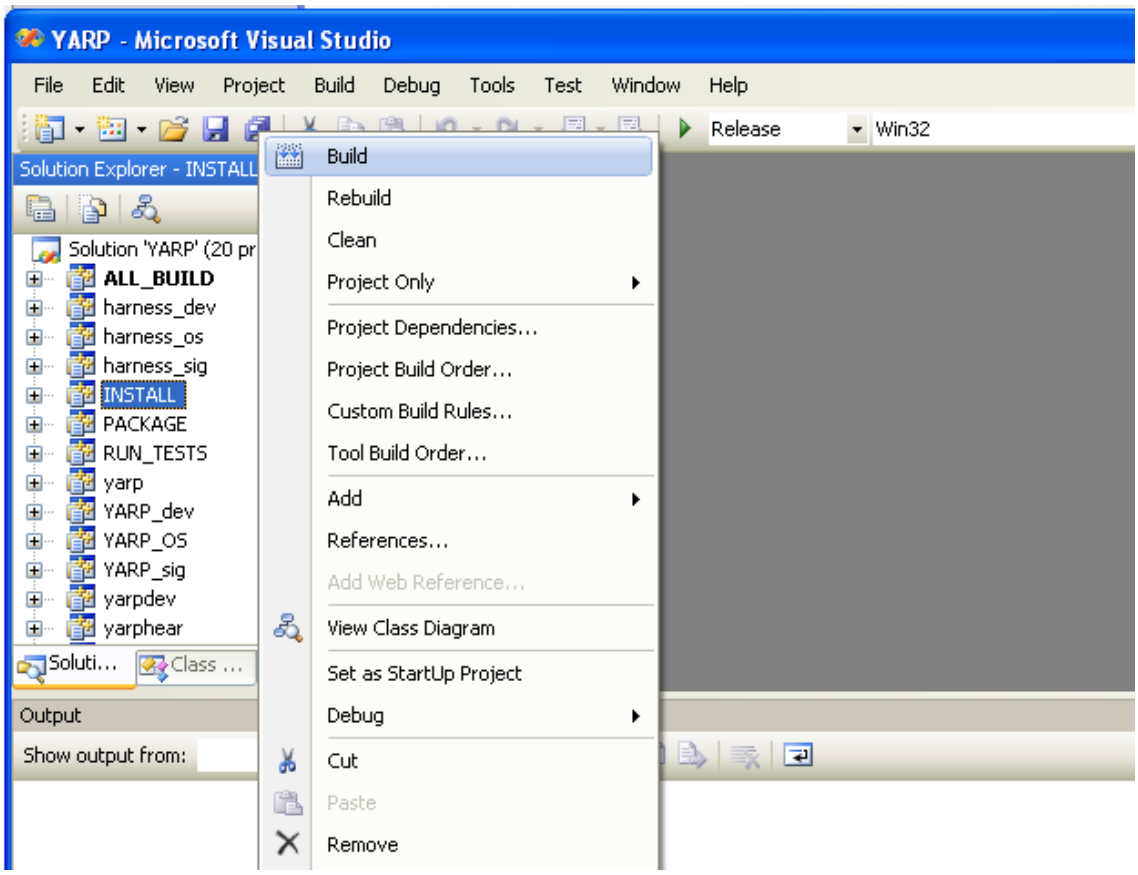

Figura 27: Instalar YARP en Visual Studio

Para crear ya el proyecto donde se va a trabajar, se descarga del repositorio la estructura básica que tienen todos lo proyecto en el ASIBOT para tener todos la misma estructura. Dentro de la carpeta anteriormente descargada del repositorio se crea una carpeta donde estará el proyecto de Visual Studio. Abrir el CMake de nuevo y en la primera opción se introduce la situación donde está la estructura del proyecto y en segundo lugar se introduce la carpeta que se ha creado y se pulsa generate.

A la hora de compilar puede haber problemas con las bibliotecas de que no las encuentra para esto hay tres opciones:

Mirar en Tools->Options->Projects and Solutions->VC++ Directories. Hay se introduce la dirección de la carpeta que tenga las bibliotecas que esté dando problemas.

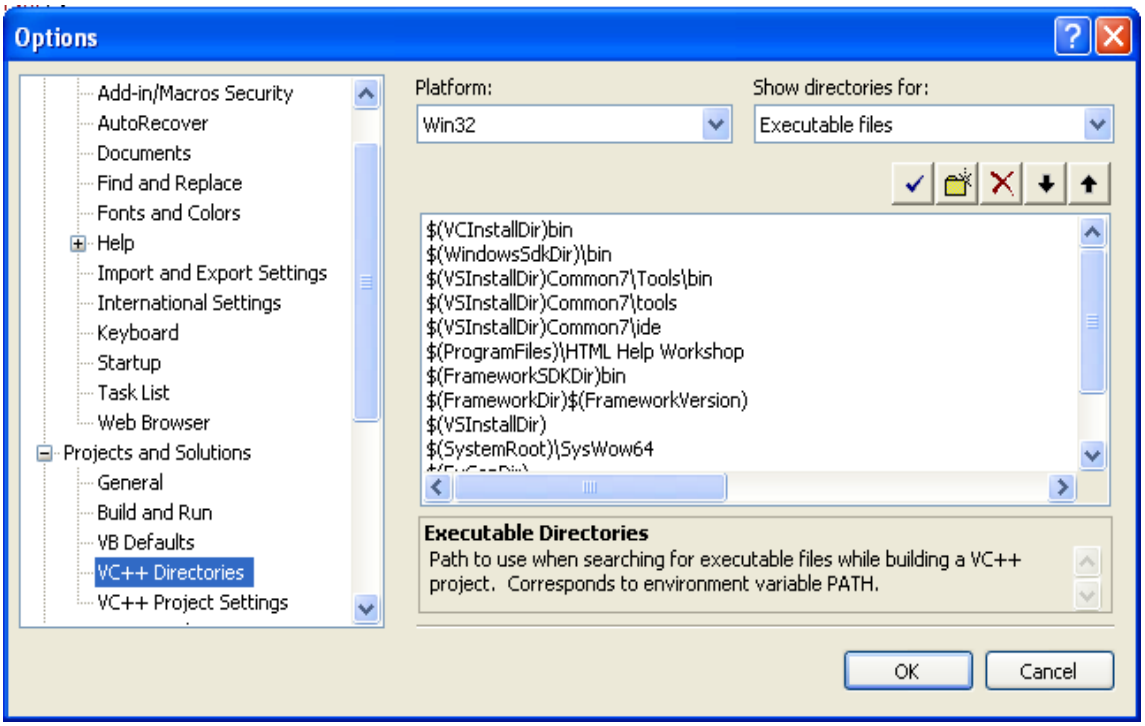

Figura 28: Introducir las bibliotecas en Visual Studio

Otra opción es dando al botón derecho encima del proyecto, se le da a propiedades ->C/C++->Additional Include Directories. Se introduce las direcciones de las carpetas donde están las cabeceras.

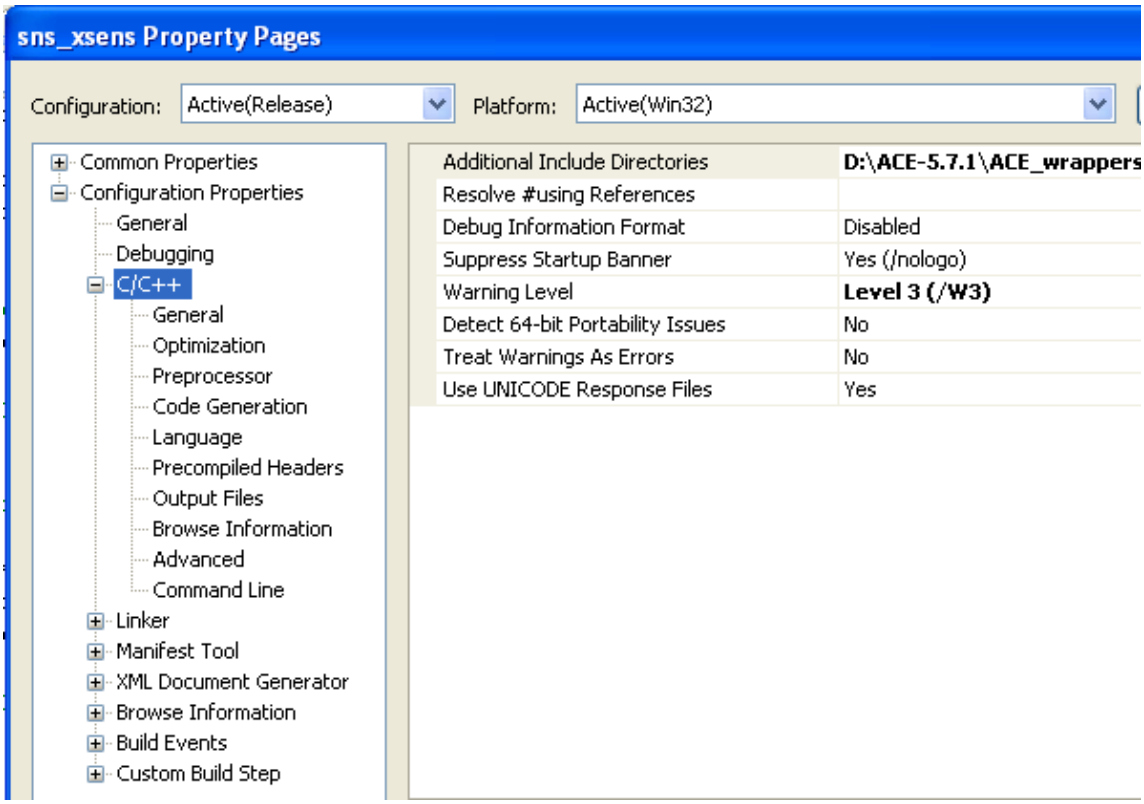

Figura 29: Lugar donde introducir las bibliotecas en Visual Studio

La otra opción es también en propiedades->Linker->Aditional Dependencies e introducimos la bibliotecas que nos fallan.

3.2 Código del componente software para robótica

En este apartado hablaremos del software del componente software para robótica sabiendo el significado de cada una de las funciones y los pasos para poner el sensor en marcha y enviarlos por YARP.

Este define es para que cuando haya un error en la puesta en marcha del sensor nos muestre por pantalla cual es el error, para ello utilizamos cmtGetResultText(res) que nos da el comentario del error.

```
#define EXIT_ON_ERROR(res,comment) if (res != XRV_OK ) { printf("Error %d occurred in " comment ": %s\n",res,cmtGetResultText(res)); exit(1); }
```

La función doHardwareScan() escanea los puertos para saber cuántos y donde están los Motion Trackers. En este se utilizan varias funciones:

- cmtScanPorts() escanea los puertos buscando los Xsens y la información la guarda en portInfo
- cmtOpenPort() abre los puertos en que tengamos Motion Trackers
- cmtGetMtCount() conseguimos el número de Motion Trackers que tenemos conectados
- cmtGetMtDeviceId() nos dice el Id de los dispositivos conectados
- cmtSetQueueMode() configura el Motion Trackers para que recibamos los datos más recientes

```
void doHardwareScan()
{
	XsensResultValue res;
	CmtPortInfo portInfo[256];
	unsigned long portCount = 0;

	printf("Buscando los dispositivos Xsens conectados...");
	res = cmtScanPorts(portInfo, &portCount, 0);
	EXIT_ON_ERROR(res,"cmtScanPorts");
	printf("done\n");
```

```
        if (portCount == 0) {
                printf("No se ha encontrado MotionTrackers \n\n");
                exit(0);
        }

        for(int i = 0; i < (int)portCount; i++) {
                printf("Usando el puerto COM %d a %d baudios\n\n",
                        (long) portInfo[i].m_portNr, portInfo[i].m_baudrate);
        }

        printf("Abriendo puerto...");
        //abre el puerto del sensor y lo pone a los baudios del sensor
        for(int p = 0; p < (int)portCount; p++){
                res = cmtOpenPort(instance, portInfo[0].m_portNr,
portInfo[0].m_baudrate);
                EXIT_ON_ERROR(res,"cmtOpenPort");
}
        printf("hecho\n\n");

        //consegir el numero de sensores MT.
        printf("Recibiendo el numero de MotionTracker \n");
        res = cmtGetMtCount(instance,&mtCount);
        EXIT_ON_ERROR(res,"cmtGetMtCount");
        printf("El numero de MotionTracker: %i\n\n",mtCount);

        // recibimos el IDs del sensor
        printf("Recibimos el ID(s)del dispositivo MotionTrackers \n");
        for(unsigned int j = 0; j < mtCount; j++ ){
                res = cmtGetMtDeviceId(instance, &deviceIds[j], j);
                EXIT_ON_ERROR(res,"cmtGetDeviceId");
                printf("ID del dispositivo %i: %08x\n",j,(long) deviceIds[j]);
```

```
        }

        // configurar para coger los datos mas recientes
        printf("\n Configuracion para que obtengamos los datos mas recientes\n\n");
        res = cmtSetQueueMode(instance,CMT_QM_LAST);
        EXIT_ON_ERROR(res,"cmtSetQueueMode");
}
```

La función doMtSettings() lo que hace es configurar el sensor en el modo de salida y a la frecuencia de muestreo que queremos. Para ello se utilizan las siguientes funciones:

- cmtGotoConfig() pone al sensor en modo configuración.
- cmtGetSampleFrequency() le indicamos la frecuencia de la salida de datos que es 100 Hz.
- cmtSetDeviceMode() pone el modo de salida en nuestro caso el modo calibrado inercial y de datos magnéticos.
- cmtGotoMeasurement() configura el sensor en modo medición para que envíe los datos como le hemos configurado con esta función.

```
void doMtSettings(void)
{
        XsensResultValue res;

        // pone al sensor a modo configuracion
        res = cmtGotoConfig(instance);
        EXIT_ON_ERROR(res, "cmtGotoConfig");
```

```
        unsigned short sampleFreq;
        res = cmtGetSampleFrequency(instance, &sampleFreq, deviceIds[0]);

        // configurar el modo de salida del sensor
        printf("Configuracion del modo de seleccion");
        for (int i=0; i < mtCount; i++) {
                if (cmtIdIsMtig(deviceIds[i])) {
                        res = cmtSetDeviceMode(instance, mode,settings, sampleFreq,
deviceIds[i]);//esto es para un dispositivo con gps
                } else {
                        res = cmtSetDeviceMode(instance, mode & 0xFF0F, settings,
sampleFreq, deviceIds[i]);
                }
                EXIT_ON_ERROR(res, "setDeviceMode");
        }

        // poner al sensor en modo medicion
        res = cmtGotoMeasurement(instance);
        EXIT_ON_ERROR(res, "cmtGotoMeasurement");
}
```

La función principal lo que hace es crear el puerto YARP para la salida de los datos con BufferedPort<Bottle> luego abre el puerto con port.open luego creamos las variables donde se guardan los datos que recibimos del sensor. Después de esto creamos la instancia y llamamos a la función doHardwareScan() donde escanea donde está el sensor, a continuación configuramos el modo de salida con doMtSettings() y creamos un bucle infinito donde con cmtGetNextDataBundle() recibe el siguiente paquete de datos y con cmtDataGetCalData() nos da los datos actuales y los enviamos por YARP a través del puerto con la función Bottle& output= port.prepare() y port.write().

```
int main() {
        Network YARP;

        BufferedPort<Bottle> port;

        port.open("/Xsens");



                double accX = 0;//inicializa una variable de coma flotante
                double accY = 0;
                double accZ = 0;
                double gyrX = 0;
                double gyrY = 0;
                double gyrZ = 0;
                double magX = 0;
                double magY = 0;
                double magZ = 0;


        XsensResultValue res = XRV_OK;

        // crea la instancia para manejar el sensor
        char serialNumber[] = KEY;
        instance = cmtCreateInstance(serialNumber);

        if (instance != -1)
                printf("Creada instancia CMT \n\n");
        else {
                printf("Creacion de la instancia CMT fallada, probablemente por un
```

```
serial number invalido\n");
                exit(1);
        }

        // escanear hardware
        doHardwareScan();

        // esperamos a resultados del escaneo
        Sleep(2000);

        // configurar la salida del sensor

        mode = CMT_OUTPUTMODE_CALIB;
        settings = 0;
        doMtSettings();

        // esperamos al primer dato
        Sleep(20);

        CmtCalData caldata;


        while(res == XRV_OK)
        {
                //obtener el paquete de datos
                res = cmtGetNextDataBundle(instance);
                Sleep(10);

                        for (unsigned int i = 0; i < mtCount; i++) {
```

```
                    if ((mode & CMT_OUTPUTMODE_CALIB) != 0) {


                    res = cmtDataGetCalData(instance, &caldata,
deviceIds[i]);


                accX=caldata.m_acc.m_data[0];
                accY=caldata.m_acc.m_data[1];
                accZ=caldata.m_acc.m_data[2];
                gyrX=caldata.m_gyr.m_data[0];
                gyrY=caldata.m_gyr.m_data[1];
                gyrZ=caldata.m_gyr.m_data[2];
                magX=caldata.m_mag.m_data[0];
                magY=caldata.m_mag.m_data[1];
                magZ=caldata.m_mag.m_data[2];


        Bottle& output = port.prepare();//output es donde se almacena los datos que
luego se envian por el puerto "port"
        output.clear();//vacia el bottle
        output.addString("accX");
        output.addDouble(accX);
        output.addString("accY");
        output.addDouble(accY);
        output.addString("accZ");
        output.addDouble(accZ);
        output.addString("gyrX");
        output.addDouble(gyrX);
        output.addString("gyrY");
        output.addDouble(gyrY);
```

```
        output.addString("gyrZ");
        output.addDouble(gyrZ);
        output.addString("magX");
        output.addDouble(magX);
        output.addString("magY");
        output.addDouble(magY);
        output.addString("magZ");
        output.addDouble(magZ);
        port.write();

                                                    }

                                }



        }

        cmtClose(instance);
        cmtDestroyInstance(instance);

 return 0;
}
```

3.3 Puesta en marcha del sensor

Para poner en marcha el sensor se tiene que seguir los siguientes pasos:

1) Abrir un terminal y poner “yarpserver” para que se puedan comunicar el sensor con los demás puertos.

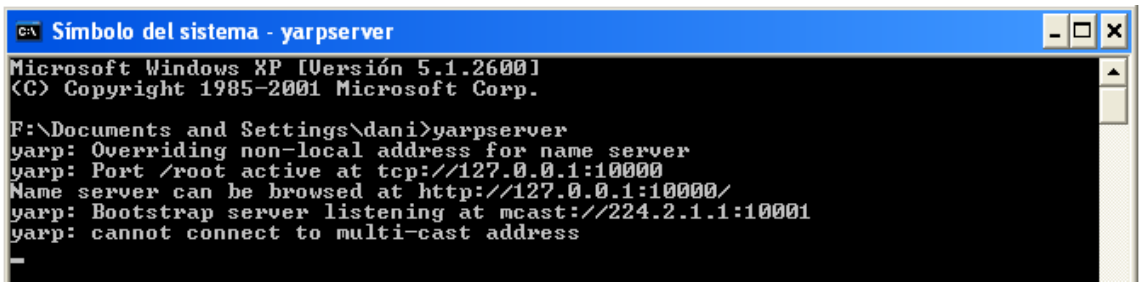

Figura 30: Terminal activando yarpserver

2) Ejecutar el componente software para robótica.

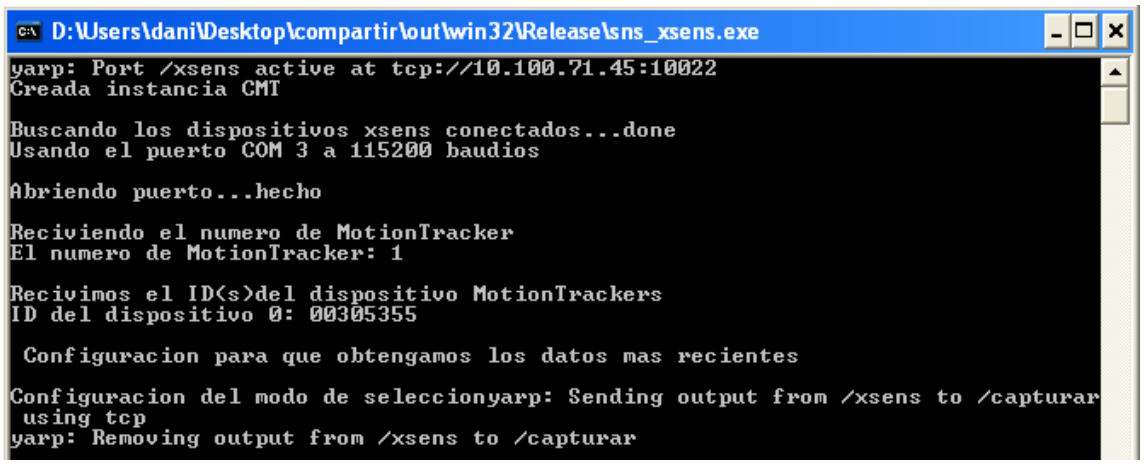

Figura 31: Terminal cuando ejecutas el programa sns_xsens

3) Abrir otro terminal donde creemos el puerto por donde se van a recibir los datos para ello ponemos lo siguiente en el terminal YARP read /leer.

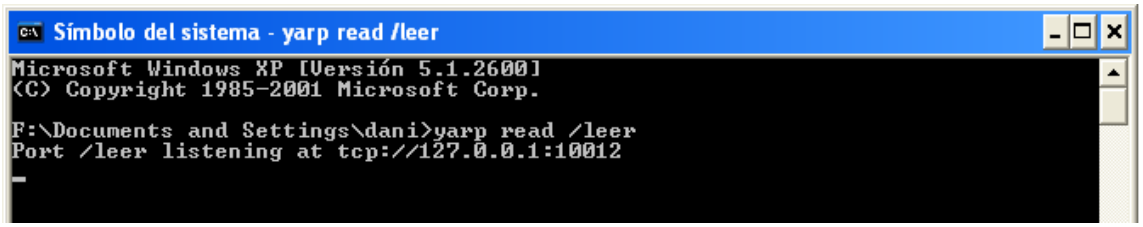

Figura 32: Terminal con puerto de lectura /leer

4) Conectamos el puerto salida del sensor con el puerto entrada del terminal leer y eso se hace en otro terminal introduciendo YARP connect /xsens /leer.

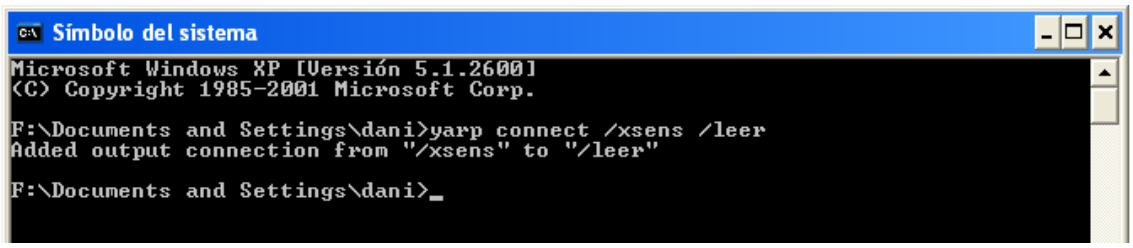

Figura 33: Terminal con conexionado de puertos /xsens y /leer

Tras estos pasos si seleccionáis el terminal donde creamos el puerto leer saldrán los datos medidos por el sensor.

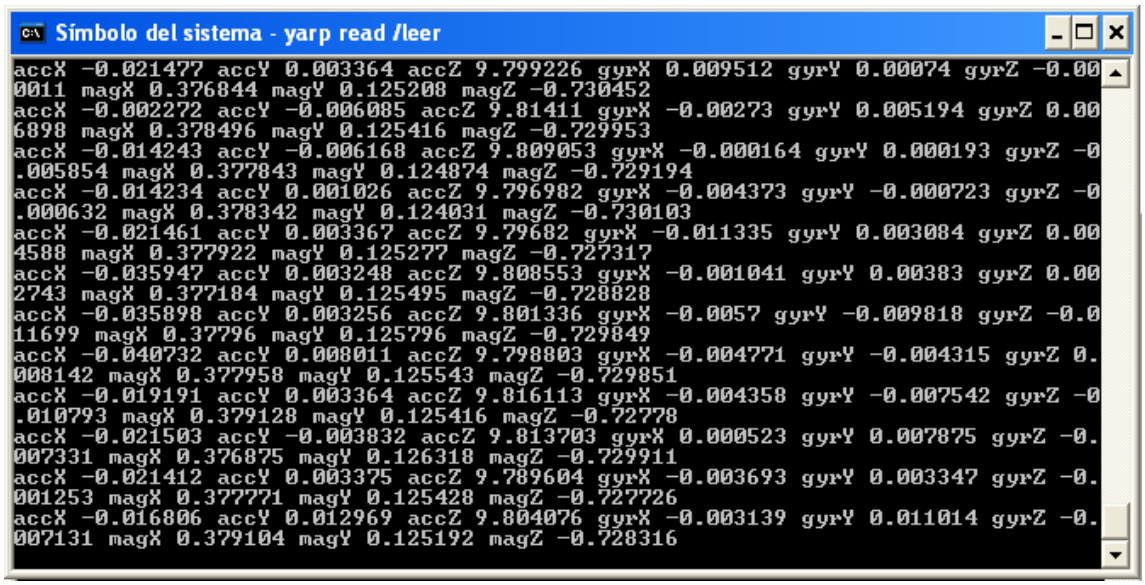

Figura 34: Terminal con los datos que recibe el puerto /leer

3.4 Código del componente software Tol_visualize

Para la mejor visualización de los datos a la hora de presentación he creado un programa para visualizar los datos a través de “yarpview” y a continuación se explica.

En este programa lo que hace en primer lugar crea 2 puertos YARP que serán, uno el que reciba del sensor y otro que envía al “yarpview” la definición de la pantalla y los puntos de los datos.

```
BufferedPort<ImageOf<PixelRgb> > port;
BufferedPort<Bottle> portImagen;
```

Después abre los puertos.

```
Property options;
options.fromCommand(argc,argv);
port.open(options.check("name",Value("/imagen")).asString());
portImagen.open("/capturar");
```

Crea las matrices que van a albergar cada uno de los datos distintos que nos da el sensor.

```
int Maccx[500];
int Maccy[500];
int Maccz[500];
int Mgyrx[500];
int Mgyry[500];
int Mgyrz[500];
int Mmagx[500];
int Mmagy[500];
int Mmagz[500];
```

Luego hay un bucle infinito donde lo primero que hace es coger los datos del paquete que se recibe y se guarda en su matriz correspondiente y en la posición indicada por la variable P. Después cogemos y enviamos la configuración de la pantalla y a continuación los puntos guardados en las matrices donde guarda 500 valores y luego los sobrescribe.

```
   while (true) {

//-------------------------------------------------------------------------------------------------
-----
```

```
//
//        FUNCION PARA CAPTAR DATOS
//
//-------------------------------------------------------------------------------------------------
-----

        Bottle *input = portImagen.read();//lo que se lee por el puerto se mete en un
objeto llamado input de la clase bottle

        if(true){

        Maccx[P] = input->get(1).asDouble();
        Maccy[P] = input->get(3).asDouble();
        Maccz[P] = input->get(5).asDouble();
        Mgyrx[P] = input->get(7).asDouble();
        Mgyry[P] = input->get(9).asDouble();
        Mgyrz[P] = input->get(11).asDouble();
        Mmagx[P] = input->get(13).asDouble();
        Mmagy[P] = input->get(15).asDouble();
        Mmagz[P] = input->get(17).asDouble();

//-------------------------------------------------------------------------------------------------
-----

      ImageOf<PixelRgb>& img = port.prepare();
      img.resize(500,500);//tamaño de la pantalla
      img.zero();//poner la pantalla a negro
      PixelRgb white(255,255,255);//define el color
                PixelRgb blue(0,0,255);//define el color
                PixelRgb green(0,255,0);//define el color
                PixelRgb red(255,0,0);//define el color
```

```
                addSegment(img,white,10, 50, 500, 50);//linea de accx
                addSegment(img,white,10, 100, 500, 100);//linea de accy
                addSegment(img,white,10, 150, 500, 150);//linea de accz
                addSegment(img,white,10, 200, 500, 200);//linea de gyrx
                addSegment(img,white,10, 250, 500, 250);//linea de gyry
                addSegment(img,white,10, 300, 500, 300);//linea de gyrz
                addSegment(img,white,10, 350, 500, 350);//linea de magx
                addSegment(img,white,10, 400, 500, 400);//linea de magy
                addSegment(img,white,10, 450, 500, 450);//linea de magz
                addSegment(img,white,10, 0, 10, 500);



for(i=0;i<500;i++){
indice=(i)%500;
                addCircle(img, blue, 10+indice, 50+Maccx[indice],  1);
                addCircle(img, red,  10+indice, 100+Maccy[indice], 1);
                addCircle(img, green,10+indice, 150+Maccz[indice], 1);
                addCircle(img, blue, 10+indice, 200+Mgyrx[indice], 1);
                addCircle(img, red,  10+indice, 250+Mgyry[indice], 1);
                addCircle(img, green,10+indice, 300+Mgyrz[indice], 1);
                addCircle(img, blue, 10+indice, 350+Mmagx[indice], 1);
                addCircle(img, red,  10+indice, 400+Mmagy[indice], 1);
                addCircle(img, green,10+indice, 450+Mmagz[indice], 1);
                }
port.write();
Time::delay(0.005);
                P = (P+1)%499;
        }
}
```

3.5 Puesta en marcha del componente software Tol_visualize

Para poner en marcha la visualización de los datos del sensor se tiene que seguir los siguientes pasos:

1) Abrir un terminal y poner “yarpserver” para que se puedan comunicar el sensor con los demás puertos.

2) Ejecutar el componente software para robótica.

3) Ejecutar el programa de la visualización.

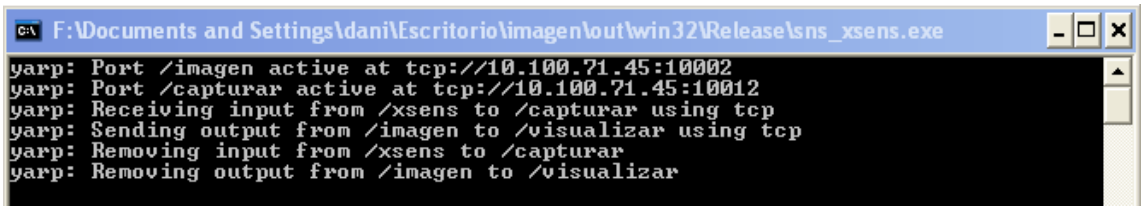

Figura 35: Terminal cuando ejecutas el programa de la visualización

4) Abrir otro terminal donde abrimos el visualizador “yarpview” que se hace de la siguiente manera “yarpview /visualizar”.

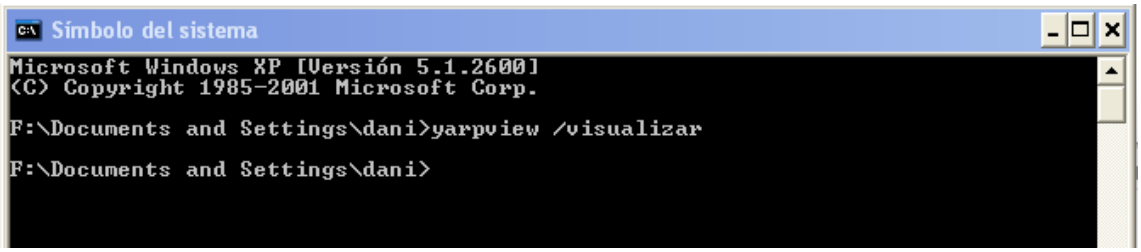

Figura 36: Terminal que ejecuta yarpview con el nombre /visualización

5) Conectamos el puerto salida del sensor con el programa realizado para la visualización con otro terminal a través de “yarp connect /xsens /capturar”.

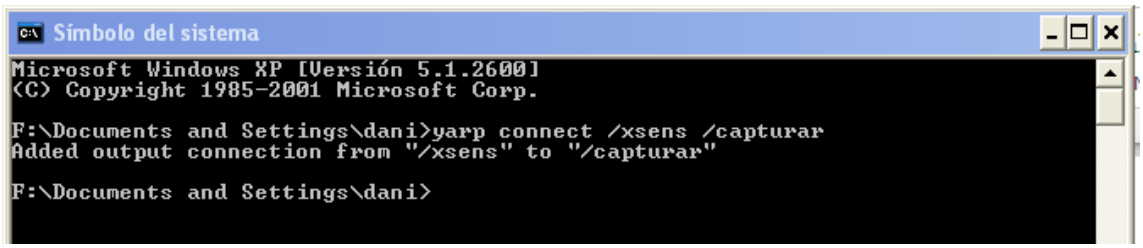

Figura 37: Terminal con conexionado de puertos /xsens y /capturar

6) Por ultimo conectamos el puerto de salida del programa de la visualización y lo conectamos a “yarpview” y eso se hace introduciendo en el terminal “yarp connect /imagen /visualizar”.

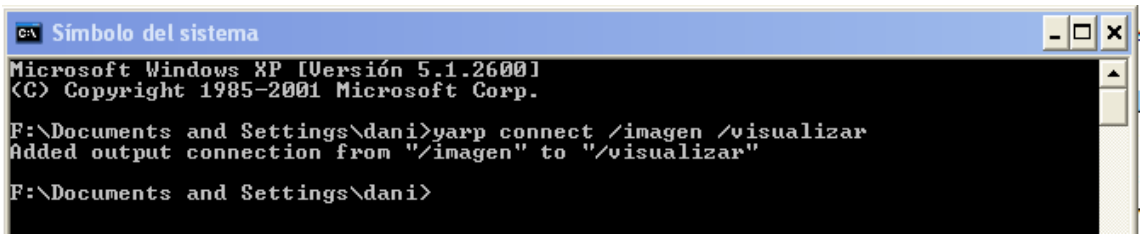

Figura 38: Terminal con conexionado de puertos /imagen y /visualizar

Tras estos pasos si seleccionáis la ventana de “yarpview” se muestran los datos en una gráfica.

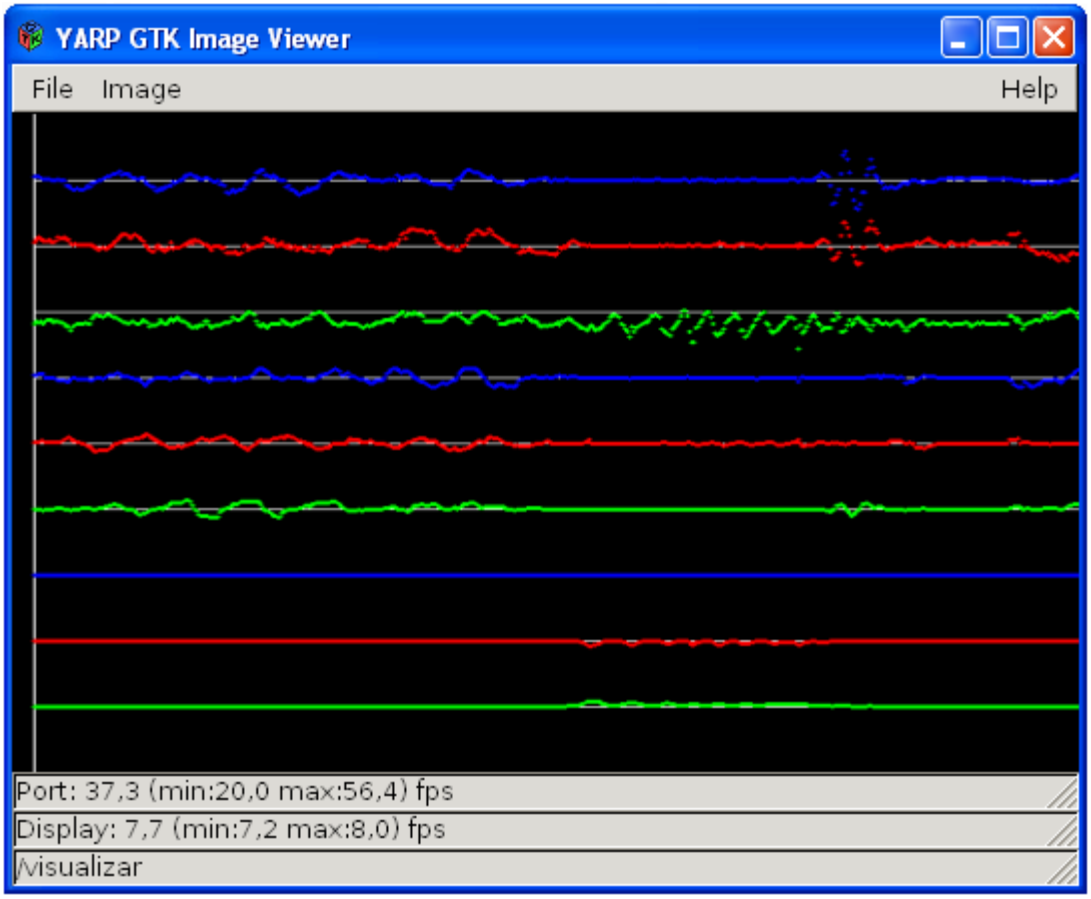

Figura 39: Yarpview con datos recibidos del sensor

4 Conclusiones

En este capítulo se hace balance de los objetivos cumplidos por el proyecto y se proponen una serie de mejoras del mismo.

4.1 Pruebas

En este punto se definen las distintas pruebas que se han realizado al componente software a lo largo del proyecto.

En primer lugar se hizo una prueba con el software que viene proporcionado por los fabricantes con el sensor, para verificar su correcto funcionamiento. Para esta prueba solo fue necesario instalar el software, conectar el sensor al puerto USB, y ejecutar el programa “MT manager”. Una vez ejecutado se elige la configuración de salida del sensor y lo muestra por pantalla en tres gráficas, una muestra la aceleración lineal, otra la velocidad angular y la última el campo magnético. En cada grafica se muestra los tres ejes, cada uno con un color como se puede ver en la figura 40.

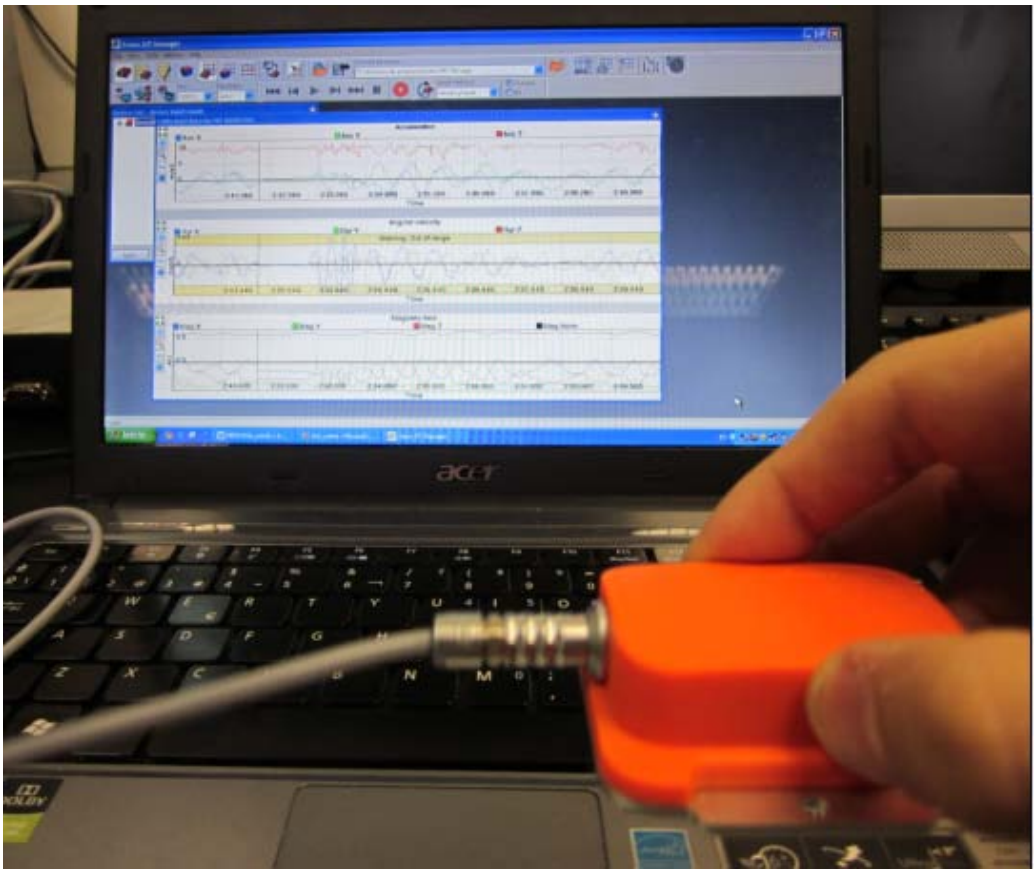

Figura 40: Prueba con el software de Xsens

En segundo lugar, tras realizar el código del componente software para robótica explicado en el punto 3.2, se hizo la puesta en marcha siguiendo los pasos indicados en el punto 3.3. En la figura 41 se puede ver cómo se muestran los datos por un terminal donde se ha instanciado un módulo lector del tipo “yarp read” y conectado al puerto de salida del módulo “Sns_Xsens”. En la prueba de la figura, esta conexión se realiza por TCP sobre la misma máquina. Sin embargo, la arquitectura software permite otras formas de conexión que se han probado, bien por memoria compartida sobre la misma máquina, o sobre TCP dentro de una red local o internet.

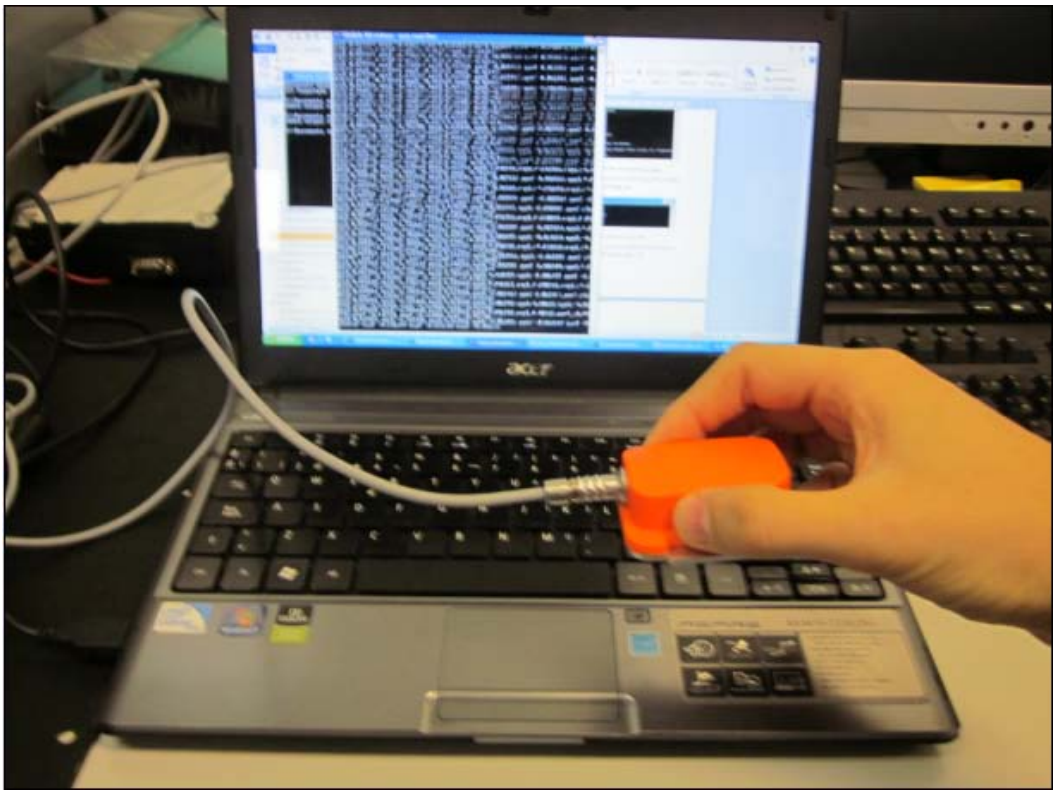

Figura 41: Prueba con salida en terminal

La tercera prueba que se hizo fue ya una vez implementado el componente software para la visualización Tol_visualize explicado en el punto 3.4, siguiendo los pasos indicados en el punto 3.5 mostrando los datos del sensor por el visor de imagen como se puede ver en la figura 42. De nuevo, estas pruebas se muestran ejecutando los componentes sobre la misma máquina, pero conservan su funcionalidad sobre diversidad de protocolos.

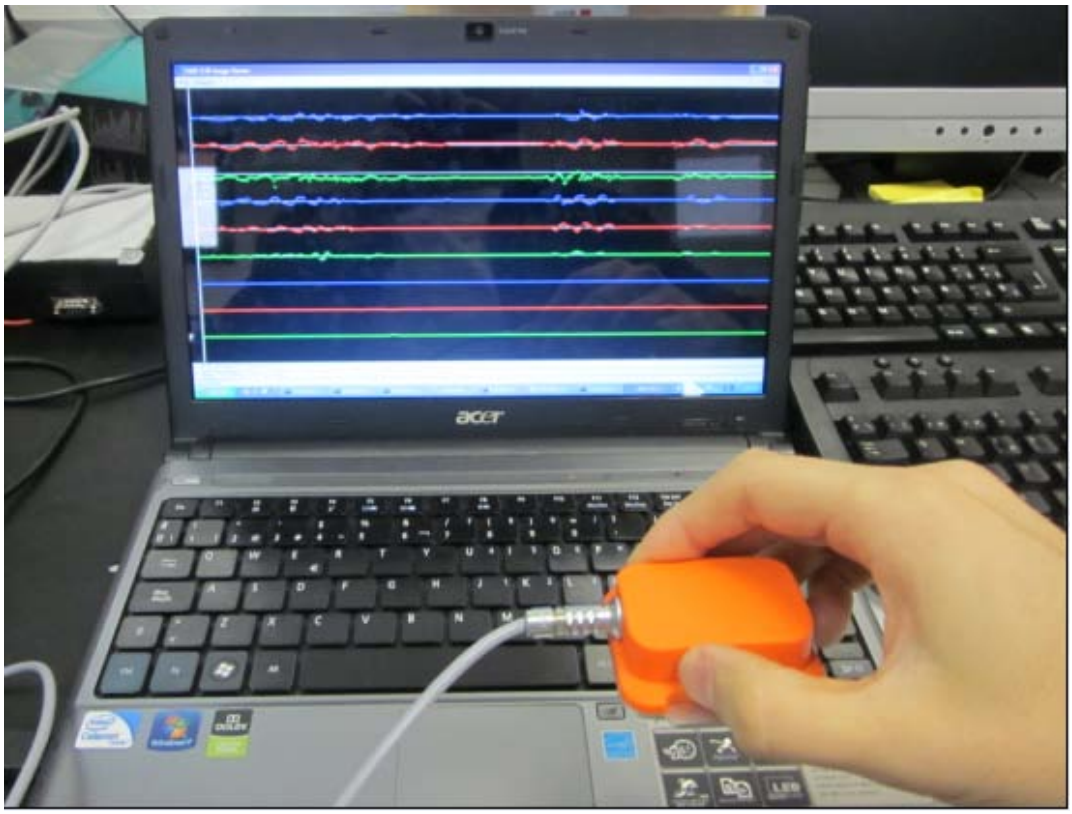

Figura 42: Prueba con salida en yarpview

Por último se hizo una prueba con el simulador del ASIBOT. Para ello se adaptó el programa para que se comunicara con los motores del simulador del ASIBOT (*jmc_rave*). El mensaje que enviaba el sensor ahora es la posición angular (roll, pitch, yaw) como se puede ver en la tabla 9, y para que se adapte al mensaje que necesita el simulador del ASIBOT se le introduce un 2 para que sepa que los valores son en modo absoluto, dos ceros indicando que no se mueven los ejes 4 y 5 de ASIBOT y al final un 100 que indica el tanto por ciento de la velocidad de los motores.

Tabla 9: Paquete enviado al simulador del ASIBOT

Significado del paquete	Movimiento absoluto	Eje1	Eje2	Eje3	Eje4	Eje5	Velocidad General (%)
Paquete enviado	2	Roll	Pitch	Yaw	0	0	100

Una vez se hizo esas adaptaciones se hizo la prueba con el simulado como se puede ver en la figura 43.

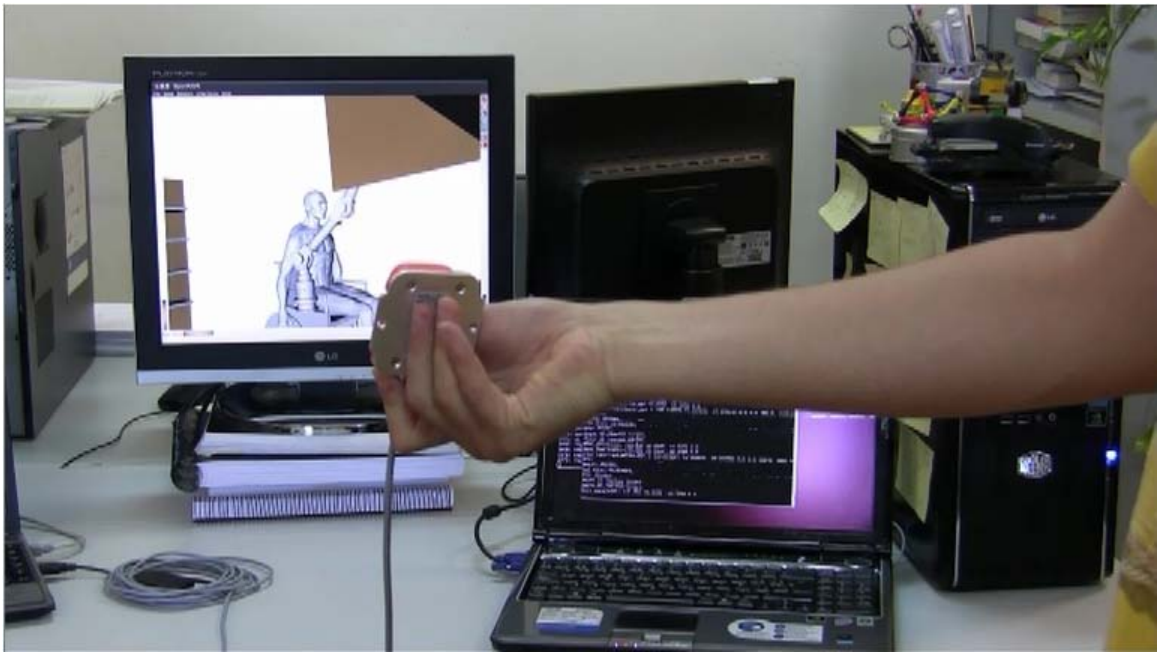

Figura 43: Prueba con el simulador del ASIBOT

4.2 Conclusiones finales

Los objetivos de este proyecto fin de carrera eran la realización del diseño y la posterior implementación del componente software para robótica. Para ello se pusieron unos objetivos intermedios que son:

- Estudiar el sensor y las funciones que te dan para implementar el componente software para robótica.
- Hacer el algoritmo para poder interactuar con el sensor y la captación de datos.
- Crear el algoritmo para poder enviar los datos al ASIBOT a través de la arquitectura software YARP.

Como ha podido comprobarse a lo largo de la lectura previa, el software no solo se ha diseñado para la implementación en el ASIBOT sino que se ha diseñado para que se

pueda poner en cualquier robot que esté integrado o se pueda integrar con la arquitectura software YARP y/o siguiendo las guías RCGv03 independientemente del sistema que utilice.

Adicionalmente, se ha creado un programa para poder ver en una gráfica los datos recogidos por el sensor para la mejor visualización.

En conclusión, se ha creado un software que hace que el sensor MTi sea un módulo funcional para sistema que tenga el robot, y además se ha puesto en marcha en modo experimental para control directo sobre ASIBOT con resultados satisfactorio.

4.3 Ampliaciones futuras

En este apartado se va hablar de las futuras ampliaciones que se pueden llevar acabo a raíz de este proyecto.

Como se ve, este proyecto es la creación de un componente software que proporciona al ASIBOT un dispositivo de control y/o manejo basado en información de inclinómetros, magnetómetros y acelerómetros. Como futuras ampliaciones se espera la creación de un programa o conjunto de programas que utilicen los datos llegados para controlar el movimiento del ASIBOT de forma inteligente.

Otra futura ampliación puede ser el estudio de la integración de varios dispositivos sensores MTi sobre robot ASIBOT y cómo tratar y fusionar los datos recibidos.

Para un mayor aprovechamiento del sensor, se puede desarrollar un programa que haga que se pueda configurar el modo de salida del sensor y así tener mayores posibilidades de utilización.

Anexos

Anexo A: Codigo fuente de componentes sofware

Anexo A.1: Código fuente del componente Sns_Xsens

```
// sns_xsens.cpp : codigo del envoltorio.

#include <yarp/os/all.h>
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include "XsensCMT.h"

void doHardwareScan();
void doMtSettings(void);



// registro del sensor
#define KEY "32DFF-5DA56-686A1-3B59A"


// test de error del sensor
#define EXIT_ON_ERROR(res,comment) if (res != XRV_OK ) { printf("Error
%d occurred in " comment ": %s\n",res,cmtGetResultText(res)); exit(1); }

long instance = -1;

CmtOutputMode mode;
CmtOutputSettings settings;
unsigned short mtCount = 0;
```

```
CmtDeviceId deviceIds[256];
CmtVector positionLLA;


using namespace std;
using namespace yarp::os;


int main() {
      Network yarp;

    BufferedPort<Bottle> port;

      port.open("/xsens");


            double accX = 0;//inicializa una variable de coma flotante
                  double accY = 0;
                  double accZ = 0;
                  double gyrX = 0;
                  double gyrY = 0;
                  double gyrZ = 0;
                  double magX = 0;
                  double magY = 0;
                  double magZ = 0;


      XsensResultValue res = XRV_OK;

      // crea la instancia para manejar el sensor
      char serialNumber[] = KEY;
      instance = cmtCreateInstance(serialNumber);

      if (instance != -1)
            printf("Creada instancia CMT \n\n");
      else {
            printf("Creacion de la instancia CMT fallada, probablemente
por un serial number invalido\n");
```

```
            exit(1);
      }

      // escanear hardware
      doHardwareScan();

      // esperamos a resultados del escaneo
      Sleep(2000);

      // configurar la salida del sensor

      mode = CMT_OUTPUTMODE_CALIB;
      settings = 0;
      doMtSettings();

      // esperamos al primer dato
      Sleep(20);

      CmtCalData caldata;

      while(res == XRV_OK)//lo nuevo de esta actualizacion
      {
            //obtener el paquete de datos
            res = cmtGetNextDataBundle(instance);
            Sleep(10);

                  for (unsigned int i = 0; i < mtCount; i++) {


                        if ((mode & CMT_OUTPUTMODE_CALIB) != 0) {

                              res = cmtDataGetCalData(instance,
&caldata, deviceIds[i]);


                  accX=caldata.m_acc.m_data[0];
                  accY=caldata.m_acc.m_data[1];
                  accZ=caldata.m_acc.m_data[2];
```

```
                    gyrX=caldata.m_gyr.m_data[0];
                    gyrY=caldata.m_gyr.m_data[1];
                    gyrZ=caldata.m_gyr.m_data[2];
                    magX=caldata.m_mag.m_data[0];
                    magY=caldata.m_mag.m_data[1];
                    magZ=caldata.m_mag.m_data[2];


            Bottle& output = port.prepare();//output es donde se
almacena los datos que luego se envian por el puerto "port"
            output.clear();//vacia el bottle
            output.addString("accX");
            output.addDouble(accX);
            output.addString("accY");
            output.addDouble(accY);
            output.addString("accZ");
            output.addDouble(accZ);
            output.addString("gyrX");
            output.addDouble(gyrX);
            output.addString("gyrY");
            output.addDouble(gyrY);
            output.addString("gyrZ");
            output.addDouble(gyrZ);
            output.addString("magX");
            output.addDouble(magX);
            output.addString("magY");
            output.addDouble(magY);
            output.addString("magZ");
            output.addDouble(magZ);
            port.write();

                                        }

                        }


     }
```

```
      cmtClose(instance);
      cmtDestroyInstance(instance);

  return 0;
}

//------------------------------------------------------------------------
------------------------
////////////////////////////////////////////////////////////////////////
//
// doHardwareScan
//
// escanea los puertos y escanea el sensor
void doHardwareScan()
{
      XsensResultValue res;
      CmtPortInfo portInfo[256];
      unsigned long portCount = 0;

      printf("Buscando los dispositivos xsens conectados...");
      res = cmtScanPorts(portInfo, &portCount, 0);
      EXIT_ON_ERROR(res,"cmtScanPorts");
      printf("done\n");

      if (portCount == 0) {
            printf("No se ha encontrado MotionTrackers \n\n");
            exit(0);
      }

      for(int i = 0; i < (int)portCount; i++) {
            printf("Usando el puerto COM %d a %d baudios\n\n",
                  (long) portInfo[i].m_portNr, portInfo[i].m_baudrate);
      }

      printf("Abriendo puerto...");
      //abre el puerto del sensor y lo pone a los baudios del sensor
      for(int p = 0; p < (int)portCount; p++){
```

```
            res = cmtOpenPort(instance, portInfo[0].m_portNr,
portInfo[0].m_baudrate);
            EXIT_ON_ERROR(res,"cmtOpenPort");
}
      printf("hecho\n\n");

      //consegir el numero de sensores MT.
      printf("Recibiendo el numero de MotionTracker \n");
      res = cmtGetMtCount(instance,&mtCount);
      EXIT_ON_ERROR(res,"cmtGetMtCount");
      printf("El numero de MotionTracker: %i\n\n",mtCount);

      // recibimos el IDs del sensor
      printf("Recibimos el ID(s)del dispositivo MotionTrackers \n");
      for(unsigned int j = 0; j < mtCount; j++ ){
            res = cmtGetMtDeviceId(instance, &deviceIds[j], j);
            EXIT_ON_ERROR(res,"cmtGetDeviceId");
            printf("ID del dispositivo %i: %08x\n",j,(long)
deviceIds[j]);
      }

      // configurar para coger los datos mas recientes
      printf("\n Configuracion para que obtengamos los datos mas
recientes\n\n");
      res = cmtSetQueueMode(instance,CMT_QM_LAST);
      EXIT_ON_ERROR(res,"cmtSetQueueMode");
}



////////////////////////////////////////////////////////////////////////
//
// doMTSettings
//
// configuracion del sensor
// Assumes initialized global MTComm class
void doMtSettings(void)
{
```

```
	XsensResultValue res;

	// pone al sensor a modo configuracion
	res = cmtGotoConfig(instance);
	EXIT_ON_ERROR(res, "cmtGotoConfig");

	unsigned short sampleFreq;
	res = cmtGetSampleFrequency(instance, &sampleFreq, deviceIds[0]);

	// configurar el modo de salida del sensor
	printf("Configuracion del modo de seleccion");
	for (int i=0; i < mtCount; i++) {
		if (cmtIdIsMtig(deviceIds[i])) {
			res = cmtSetDeviceMode(instance, mode,settings,
sampleFreq, deviceIds[i]);//esto es para un dispositivo con gps
		} else {
			res = cmtSetDeviceMode(instance, mode & 0xFF0F,
settings, sampleFreq, deviceIds[i]);
		}
		EXIT_ON_ERROR(res, "setDeviceMode");
	}

	// poner al sensor en modo medicion
	res = cmtGotoMeasurement(instance);
	EXIT_ON_ERROR(res, "cmtGotoMeasurement");
}
```

Anexo A.2: Código fuente del componente software Tol_visualize

```
#include <ace/config.h>
#include <yarp/os/all.h>
#include <yarp/sig/all.h>
#include <math.h>

using namespace yarp::os;
using namespace yarp::sig;
using namespace yarp::sig::draw;

int main(int argc, char *argv[]) {
    Network yarp;

    BufferedPort<ImageOf<PixelRgb> > port;
      BufferedPort<Bottle> portImagen;

    Property options;
    options.fromCommand(argc,argv);
    port.open(options.check("name",Value("/imagen")).asString());
      portImagen.open("/capturar");

      int P=0;
      int Maccx[500];
      int Maccy[500];
      int Maccz[500];
      int Mgyrx[500];
      int Mgyry[500];
      int Mgyrz[500];
      int Mmagx[500];
      int Mmagy[500];
      int Mmagz[500];

      int i;
      int indice;

    while (true) {
```

```
//-----------------------------------------------------------------------
----------------------------------------
//
//    FUNCION PARA CAPTAR DATOS
//
//-----------------------------------------------------------------------
----------------------------------------

      Bottle *input = portImagen.read();//lo que se lee por el puerto se
mete en un objeto llamado input de la clase bottle

      if(true){
      //if (Bottle hasChanged()==true){

      Maccx[P] = input->get(1).asDouble();
      Maccy[P] = input->get(3).asDouble();
      Maccz[P] = input->get(5).asDouble();
      Mgyrx[P] = input->get(7).asDouble();
      Mgyry[P] = input->get(9).asDouble();
      Mgyrz[P] = input->get(11).asDouble();
      Mmagx[P] = input->get(13).asDouble();
      Mmagy[P] = input->get(15).asDouble();
      Mmagz[P] = input->get(17).asDouble();

//-----------------------------------------------------------------------
----------------------------------------

        ImageOf<PixelRgb>& img = port.prepare();
        img.resize(500,500);//tamaño de la pantalla
        img.zero();//poner la pantalla a negro
        PixelRgb white(255,255,255);//define el color
            PixelRgb blue(0,0,255);//define el color
            PixelRgb green(0,255,0);//define el color
            PixelRgb red(255,0,0);//define el color
            addSegment(img,white,10, 50, 500, 50);//linea de accx
            addSegment(img,white,10, 100, 500, 100);//linea de accy
            addSegment(img,white,10, 150, 500, 150);//linea de accz
```

```
            addSegment(img,white,10, 200, 500, 200);//linea de gyrx
            addSegment(img,white,10, 250, 500, 250);//linea de gyry
            addSegment(img,white,10, 300, 500, 300);//linea de gyrz
            addSegment(img,white,10, 350, 500, 350);//linea de magx
            addSegment(img,white,10, 400, 500, 400);//linea de magy
            addSegment(img,white,10, 450, 500, 450);//linea de magz
            addSegment(img,white,10, 0, 10, 500);


for(i=0;i<500;i++){
indice=(i)%500;
            addCircle(img, blue, 10+indice, 50+Maccx[indice],  1);
            addCircle(img, red,  10+indice, 100+Maccy[indice], 1);
            addCircle(img, green,10+indice, 150+Maccz[indice], 1);
            addCircle(img, blue, 10+indice, 200+Mgyrx[indice], 1);
            addCircle(img, red,  10+indice, 250+Mgyry[indice], 1);
            addCircle(img, green,10+indice, 300+Mgyrz[indice], 1);
            addCircle(img, blue, 10+indice, 350+Mmagx[indice], 1);
            addCircle(img, red,  10+indice, 400+Mmagy[indice], 1);
            addCircle(img, green,10+indice, 450+Mmagz[indice], 1);
            }
port.write();
Time::delay(0.005);
            P = (P+1)%499;
    }
      }
    return 0;
}
```

Anexo B: Hojas de características

Datasheet: MTi Xsens

	MTi-28A##G##	MTi-48A##G##	MTi-68A##G##
Communication interface:	Serial digital (RS-232)	Serial digital (RS-485)	Serial digital (RS-422)
Additional interfaces:	SyncIn SyncOut Analog In	SyncIn SyncOut	SyncIn
Operating voltage[22]:	4.5-30 V	4.5-30 V	4.5-30 V
Power consumption[23]: (AHRS/3D orientation mode)	350 mW	350 mW	350 mW
Temperature Operating Range:	-20°C - 55°C	-20°C - 55°C	-20°C - 55°C
Specified performance Operating Range:	0°C - 55°C	0°C - 55°C	0°C - 55°C
Outline Dimensions:	58 x 58 x 22 mm (W x L x H)	58 x 58 x 22 mm (W x L x H)	58 x 58 x 22 mm (W x L x H)
Weight:	50 g	50 g	50 g

Vector	Unit
Acceleration	m/s^2
Angular velocity (rate of turn)	rad/s
Magnetic field	a.u. (arbitrary units) normalized to earth field strength

MTi and MTx Sensor Fact Table	
Accelerometers	MEMS solid state, capacitative readout
Rate of turn sensor (rate gyroscope)	MEMS solid state, monolithic, beam structure, capacitative readout
Magnetometer	Thin film magnetoresistive

		rate of turn	acceleration	magnetic field	temperature
Unit		[deg/s]	[m/s^2]	[mGauss]	[°C]
Dimensions		3 axes	3 axes	3 axes	-
Full Scale	[units]	+/- 300*	+/- 50	+/- 750	-55...+125
Linearity	[% of FS]	0.1	0.2	0.2	<1
Bias stability	[units 1σ][11]	1	0.02	0.1	0.5[12]
Scale factor stability	[% 1σ][11]	-	0.03	0.5	-
Noise density	[units /√Hz]	0.05[13]	0.002	0.5 (1σ)[14]	-
Alignment error[(15)]	[deg]	0.1	0.1	0.1	-
Bandwidth	[Hz]	40	30	10	-
A/D resolution	[bits]	16	16	16	12

Bibliografía

[1] Correal Tezanos, ‘Diseño e implementación de la arquitectura de control del Robot Asistencial MATS’. Universidad Nacional de Educación a distancia, 2005.

[2] Víctor Placer De Miguel, ‘Implementación de las comunicaciones internas mediante CANbus en el robot ASIBOT_V1.5’. Universidad Carlos III de Madrid, 2008.

[3] Raúl Palencia López, ‘Migración de la plataforma a bordo del robot asistencial ASIBOT’. Universidad Carlos III de Madrid, 2009.

[4] Alberto Jardón Huete, ‘Metodología de diseño de robots asistenciales. Aplicación al robot portátil ASIBOT’. Universidad Carlos III de Madrid, 2006.

[5] Giménez Fernández A. ‘Metodología de Diseño y Control de Robots Escaladores. Aplicación a las Tareas de Inspección’. Universidad Carlos III de Madrid, 2000.

[6] Juan G. Victores, ‘Software engineering techniques applied to assistive robotics’. Universidad Carlos III de Madrid, 2010.

[7] http://robots.uc3m.es/w/index.php/Main_Page. “Visitado el día 20 de noviembre de 2010”

[8] http://eris.liralab.it/yarpdoc/index.html. “Visitado el día 28 de septiembre de 2010”